USENIX Association

# Proceedings of SRUTI '05

## Steps to Reducing Unwanted Traffic on the Internet Workshop

July 7, 2005
Cambridge, MA, USA

# Program Organizers

## Program Chairs

Dina Katabi, *MIT*
Balachander Krishnamurthy, *AT&T Labs—Research*

## Program Committee

Paul Barford, *University of Wisconsin*
Steven M. Bellovin, *Columbia University*
Herve Debar, *France Telecom R&D*
Mark Handley, *University College London*
Doug Maughan, *U.S. Department of Homeland Security*
Chris Morrow, *UUNET*
Vern Paxson, *ICIR/ICSI*
Dawn Song, *Carnegie Mellon University*
Paul Vixie, *ISC*

## Steering Committee

Clem Cole, Ammasso, *USENIX Liaison*
Dina Katabi, *MIT*
Balachander Krishnamurthy, *AT&T Labs—Research*

## The USENIX Association Staff

# External Reviewer

Adam Greenhalgh

# SRUTI '05
# Steps to Reducing Unwanted Traffic on the Internet Workshop
## Cambridge, MA, USA



## Thursday, July 7, 2005

**DDoS and Worms**
*Session Chair: Vern Paxson, ICIR/ICSI*



**Spam-1**
*Session Chair: Paul Vixie, ISC*



**Bots and Spoofed Sources**
*Session Chair: Chris Morrow, UUNET*



**Adaptive Defense Systems**
*Session Chair: Dina Katabi, MIT*

**Spam-2 and Encryption**

*Session Chair: Steve Bellovin, Columbia University*

# Index of Authors

# Message from the Program Co-Chairs

Hello and welcome to SRUTI '05, the first workshop on unwanted traffic on the Internet. SRUTI has a broad agenda that covers attacks on many protocols (e.g., IP, TCP, DNS, BGP, and HTTP) and popular applications (e.g., email, Web), and focuses on architectural cross-layer solutions. Traditionally, solutions to the unwanted traffic problem from the research community, commercial enterprises, and backbone operators are discussed in different venues. SRUTI brings academics and corporate researchers together with those who face the problem at the operational level to look for cross-fertilization of solutions to one of the most serious problems facing the Internet.

The handiwork of enthusiastic Steering Committee members alone is not enough to bring such a task to fruition. USENIX readily agreed to sponsor the workshop. Our warm thanks to the excellent Program Committee, who reviewed the thirty-five submitted papers and, after a spirited PC meeting, selected thirteen papers. Thanks are also due to the extraordinarily professional and incredibly responsive USENIX employees who nimbly handled a variety of tasks, including Peter Collinson, Jane-Ellen Long, Tara Mulligan, and Jennifer Joost. Ellie Young trusted us and gave us the necessary leeway early on.

We are delighted that even as a first-time workshop we were able to obtain sponsorship from AT&T, Cisco Systems, and the Department of Homeland Security. Their generous sponsorship allowed us to have a nominal registration fee and helped pay for several student travel grants.

SRUTI (pronounced "shhruthi," Sanskrit for "that which is heard") introduces the role of a discussant: the Session Chair makes a brief presentation at the end of each session summarizing the work presented, other ongoing work in the area, and facilitates discussion.

We have several interesting papers on a variety of topics: DDoS, worms, bots and spoofed sources, adaptive defense systems, spam, and privacy. Papers ranged from architectural novel Internet designs through practical solutions to a particular aspect of unwanted traffic. Cutting-edge topics like spam in VoIP and the prevalence of botnets are likely to be of interest to many.

Once again, welcome to SRUTI and we hope you hear stimulating ideas.

**Dina Katabi, MIT**
**Balachander Krishnamurthy, AT&T Labs—Research**
**Program Co-Chairs**

# Using Routing and Tunneling to Combat DoS Attacks

Adam Greenhalgh, Mark Handley, Felipe Huici
*Dept. of Computer Science*
*University College London*
*Gower Street, London, WC1E 6BT*
[a.greenhalgh|m.handley|f.huici]@cs.ucl.ac.uk

## Abstract

Thorough defense against DoS attacks is extremely difficult without incurring significant changes to the Internet architecture. We present a series of changes aimed at establishing protection boundaries to reduce the effectiveness of most flooding DoS attacks against servers. Only minimal and local changes are required to current network architectures. We show that our scheme is highly beneficial even if deployed at a single ISP, with additional benefits arising from multiple-ISP deployment. Finally, we show that the changes can be implemented with off-the-shelf components.

## 1 Introduction

Denial-of-Service (DoS) attacks have become a serious threat to the future potential of the Internet. If the Internet is to become critical infrastructure, as advocated by Internet telephony and digital convergence proponents, then it must be robust to attack. This is not to say that it must be impossible to disrupt Internet systems, but that the probability of *persistent* disruption must be low. In practice, this means deterring attack through legal means, preventing attack by removing attack vectors, or enabling effective rapid response in face of attack. In reality, all three of these are likely to be needed.

In previous work[3], we examined possible changes to the Internet architecture aimed at increasing robustness. The goal was deliberately radical: to restrict the modes of operation of the network to those that are actually desired, greatly reducing the potential attack surface. We proposed separating the IP address space into "client" and "server" spaces, allowing communication between the two spaces, but not within each space, and we advocated path-based addresses for clients, rather than globally unique addresses.

The potential benefits would be significant: source address spoofing and reflection attacks would be eliminated, and worms would find it much more difficult to spread. Without spoofed addresses, reactive response mechanisms could be automated simply and safely.

Unfortunately the proposal required IPv6 to encode path-based addresses, HIP[7] to provide a stable end-system binding, and new inter-domain path-encoders to build path addresses. Peer-to-peer applications presented additional complexity. Even if there were global agreement on such an architectural change, implementation could not happen overnight.

Such practical constraints lead us to wonder how many of these potential benefits could be obtained without change to the end-systems, and with only minimal and local changes to the routing infrastructure.

The issue of local change is important. Altruism is rarely an important guiding principle for ISPs, so any changes must provide short-term benefits. More customers must be attracted, more money must be received from existing customers, or running costs must be lower if a change is to be made. Changes to the client-side of the Internet are likely to require very widespread deployment before benefits are seen by servers, and so they are unlikely to be deployed. The question then is whether there is a server-side-first deployment strategy that leads in the long run to a more robust Internet, has benefits for early adopters, and incremental benefits as more adopters come on board.

In this paper we will explore just such a strategy. The plan we will sketch out does not have all the benefits of our original architecture; there are, nonetheless, benefits at each step along the path for the ISPs doing the deployment. Most importantly, the solutions are still very general, so they do not trade off short-term expediency against the future ability to evolve the applications running on the Internet.

## 2 The Nature of the Problem

DoS attacks can target scarce resources at any layer in the stack, but in this paper we only address flooding attacks that target the IP and transport layers (e.g. TCP, UDP, DNS). As such we only present an IP layer solution rather than using an application layer overlay like the one presented by [5]. Attacks targeting higher-layer functionality are still important, but robust applications are of little use if data cannot reach them. Before examining solutions though, it is important to step back and evaluate why this is a hard problem.

There are a number of factors that complicate the picture. The simplest DoS attack consists of one host flooding traffic to another. If the recipient cannot keep up due to CPU, memory, network bandwidth, or other resource starvation, then the attack is likely to be successful. The obvious question is: why can't the recipient simply request that the traffic stop? The reasons are many and varied, and shed light on the nature of the problem.

The natural expectation is that an end-system could say to the network "don't send me any more traffic from X". There are essentially two technical issues: where in the network this would be enforced, and how to ensure that this mechanism can only be used legitimately.

In the existing Internet, there are not many places where such packet filters could be installed. To be effective, filtering must be installed upstream of any bottleneck on the path to the recipient, and ideally should be close to the source of the attack. Unfortunately, tracing back the path taken from the attacker to the victim is not trivial, given that wide-area Internet paths are almost always asymmetric. As a result, to automatically find an appropriate place to install such a filter requires mechanisms that do not currently exist. Even if such mechanisms did exist, it is far from obvious what incentive the ISP at the appropriate place for a filter would have for installing it, since an ISP close to the attacker is unlikely to have a direct business relationship with the victim. Finally there is the issue of authenticating that the request to install such a filter really did come from the victim. No appropriate authentication infrastructure currently exists, and without good authentication, any process that automates the installation of filters may be exploited as a DoS mechanism in its own right.

All these problems are made worse by *Distributed* Denial-of-Service attacks (DDoS), where many hosts are compromised and flood a victim in unison. With DDoS it becomes very important that any filters are as far upstream as possible. The sheer number of filters that the victim might need to be installed could also be a problem with existing hardware.

Any automated filter mechanism that installs filters for a specific {source, destination} pair of IP addresses will be useless against an attacker that spoofs the source address of packets. Not only can the attacker simply change address to avoid existing filters, but he might be able to spoof a legitimate machine's source address in "attack" packets, with the goal of causing a filter to be installed that prevents legitimate communication.

Recent data[2] from security vendors indicates that source address spoofing is rarely used today in DDoS attacks since it is not needed for them to be effective. Botnets are relatively easy to create, and since there is no automatic way to silence compromised hosts, there is little incentive to spoof. In fact, just enough edge-sites employ ingress filtering on outgoing traffic that a botnet actually has slightly greater firepower if address spoofing is *not* used. However if an automatic filtering mechanism were deployed, the prevalence of address spoofing would no doubt increase significantly. At present, an interesting vicious cycle exists:

1. Attacks do not source address spoof because there is no automated filtering mechanism.
2. There is relatively little deployment of ingress filtering, because attacks do not source address spoof.
3. There is no automated filtering mechanism because attacks could source-address spoof to bypass it.

One way to break such a cycle would be to deploy a mechanism to effectively defend against non-spoofed DoS attacks, while not being vulnerable to abuse by spoofed traffic. Such a mechanism would not *by itself* defend against spoofed traffic, but it would shift the balance. There would be more incentive for ingress filtering, although this seems unlikely to ever be ubiquitous. Most importantly, it opens the door for additional mechanisms to be designed that can detect the difference between spoofed and non-spoofed traffic.

In this paper we sketch out one such solution. It is not intended to be a panacea, but rather to raise the bar somewhat, and to do so with off-the-shelf technologies.

## 2.1 Points of Control

There are two important points in defending against DoS: the *point of detection* of an attack, and the *point of control* where the effects of an attack can be mitigated.

The process of attack detection is not simple. Some attacks such as SYN-flooding are easy to detect, at least at the recipient. Other attacks such as making a large number of HTTP requests to a web server are harder, since they require knowledge of what normal traffic looks like, and the purpose normally served by that traffic. Detection, however, is not the focus of this paper. Products from vendors such as Arbor Networks are available that can perform this sort of analysis, so we will assume that a detection infrastructure exists at important servers.

Our focus is on establishing *points of control*, once an attack has been detected. The current Internet has no real concept of a point of control, beyond what can be done using traditional firewalls and routing protocols. Generally firewalls are too close to the victim to be useful against DoS, and destination-based routing does not provide sufficient discrimination of traffic flows.

The requirement is for a victim to be able to communicate with a point of control upstream of local bottleneck links, and to be able to request instantiation of filters that prevent hostile traffic reaching its subnet. Such filters must be installed quickly, without human inter-

vention. In addition, the point of control must be able to validate the filter request, so filters cannot be installed maliciously by third parties.

None of this is terribly radical, and what remains is engineering rather than science. How can the victim rapidly and reliably discover the existence of a point of control on the path from the attacker? How can it do this when there are many thousands of attackers? How can the point of control validate the filter request?

Our solution answers these questions through segmentation of the address space, careful control of routing information, and encapsulation. These are the three key IP-level building blocks we have to work with in the Internet today.

## 3 Proposed Solution

Ideally we would like to protect all Internet hosts, but realistically it is usually servers that present the biggest target. We propose, consequently, to provide a protected virtual net for those servers that wish to be better defended. An ISP providing such protection could charge a premium fee for the service, giving a clear incentive for deployment. In [10] the author presents a similar approach that for small DDoS attacks in the context of a single ISP. Our solution is intended to extend to a network of many collaborating ISPs, but in this section we first examine how our solution might initially be deployed at a single ISP, before examining options how multiple ISPs might cooperate.

### 3.1 Single ISP Architecture

The first step is to designate certain subnets of the IP address space as server subnets: these will receive additional protection from attack. We refer to these subnets collectively as the *server-net*; conceptually they are within a protection boundary ringed by control points. Traffic from the public Internet must traverse one of these control points on its way into the server-net.

A condition of being a server-net host is not being permitted to send directly to other server-net hosts. This constraint prevents hosts inside the server-net from attacking other hosts inside the server-net, and prevents server-net hosts being exploited as relays in reflection attacks on other server-net hosts. It also helps slow the spread of worms within the server-net boundary.

The basic functions of a server-net boundary control point are *encapsulation* and *filtering*. At an encapsulator, packets destined for a server are encapsulated IP-in-IP, and sent to a decapsulator located in the server's co-location facility. An ISP must have at least one encapsulator, but maximum benefit will be gained with one encapsulator associated with each PoP or peering link.

The principal advantage of this architecture is that when a server is attacked, the decapsulator knows precisely which encapsulators the malicious traffic traversed. As a result, it can ask them to filter traffic, stopping the attack some distance upstream of the victim. We note that encapsulation isn't the only technique by which this could be achieved. In particular, MPLS tunneling might also be used for this purpose. However IP-in-IP encapsulation has advantages over MPLS. First, the address of the encapsulator can be directly obtained by the decapsulator, rather than needing additional mechanisms to reverse map the MPLS labels, but perhaps more importantly it is much harder to extend an MPLS solution inter-domain, which is our eventual goal.

Causing incoming traffic, even traffic originating from within the local ISP, to traverse an encapsulator requires careful control of routing. Routes to the server-net subnets should only be advertised from the encapsulators themselves, to ensure that there is no way to bypass them and send directly to the servers. In addition, the decapsulator addresses should be taken from the ISP's infrastructure address space, which (according to best current practice) should never be advertised outside of the ISP's own network. This prevents an attacker directly flooding a decapsulator associated with a server.

Possible communication paths within this architecture are illustrated in Figure 1. Flows 1 and 2 show the typical scenario with packets from a client passing through an encapsulator, being tunneled to a decapsulator, and finally arriving at the servers. Flow 3 is client to client and is unaffected. Flow 4, from one protected server to another, is disallowed to prevent reflection attacks and the spread of worms. Finally, flow 5 shows a protected server choosing to perform decapsulation itself.

Server→client traffic could be sent via the reverse of the incoming tunnel, or it could be forwarded natively. Either reverse path for the traffic is feasible with the proposed architecture, it should be an operation decision as to which is used. Tunneling has the benefit that a smart encapsulator can view both directions of a flow, allowing it to monitor and filter traffic more intelligently. However, this requires forwarding state at the decapsulator that is set up based on observed incoming traffic, and this state might be vulnerable to DoS if source address spoofing is used.

One solution is for the server to switch dynamically from a native to a tunneled reverse path once a connection is fully established and is therefore known not to be spoofed. Traffic with a tunneled reverse path can then be forwarded from encapsulator to decapsulator with higher diffserv priority, which might lessen the effect of flooding attacks that spoof source addresses.

The goal of a server-net when deployed at a single ISP is to allow automated filtering of unwanted traffic at the ingress point of that ISP. To perform such automated

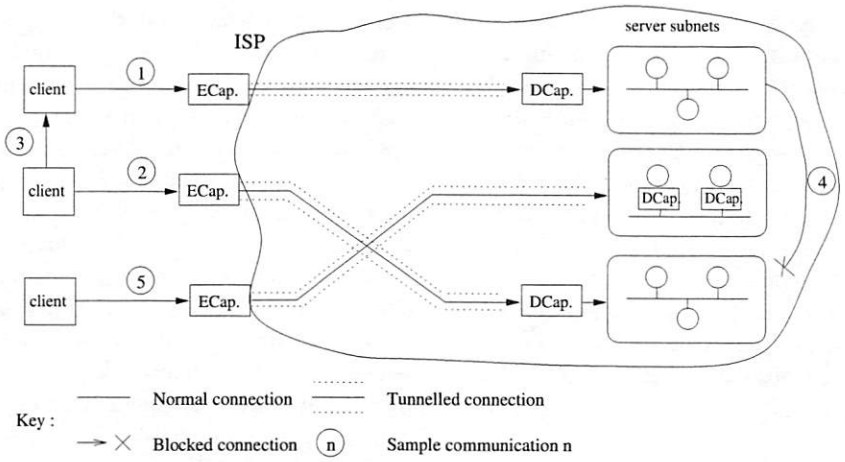



Figure 1: Scenarios for single ISP architecture.

filtering requires a detection infrastructure in place, located so that it can monitor traffic to the server. Possible locations for this would be in the decapsulator, in the server, or on the path between the two.

Once a flow has been identified as hostile, the decapsulator needs to be informed, and it in turn informs the encapsulator, which installs the appropriate filter. The use of infrastructure addresses between decapsulator and encapsulator is the first line of defense against subversion of the filtering capabilities, as it should simply not be possible for normal Internet hosts to send filtering requests directly to an encapsulator. Behind this first line of defense, the signaling channel between the decapsulator and the encapsulator should be secured. Simple nonce exchange may be sufficient to protect against off-path attacks, given that a compromised router on the path can already cause DoS. Public-key-based solutions are, of course, also possible.

The benefits are clearly greatest for large ISPs hosting server farms and running a large number of encapsulators. Such ISPs will typically have many peering points with other large ISPs, and these peering points will be both geographically and topologically distributed. This traffic from a large distributed attack will be spread across many encapsulators because it will be entering the network from many neighboring ISPs, and so it will be stopped closer to its origin, before it has aggregated to the point where it can cause serious damage.

Smaller ISPs still benefit because their customers can control their degree of exposure, but a large enough attack is likely to overwhelm all incoming links. Nothing an ISP can do *by itself* will help in such circumstances.

## 3.2 Inter-ISP communications

The benefits of a server-net increase as ISPs co-operate. Extending the protection boundary of the server-net to include the server-nets of co-operating ISPs moves the control points nearer to the sources of the DoS traffic. As we extend the protection boundary, distributed attacks become less concentrated at any particular control point, since traffic from each attacking host enters the server-net through its local encapsulator. Traffic that would previously have traversed the peering link uncontrolled now traverses the peering link encapsulated, within the server-net control boundary.

The general idea is that traffic enters the server-net at the server-net ISP closest to the traffic source, and is then tunneled from that ISP's encapsulator directly to a decapsulator at the destination ISP border. At the destination ISP, the traffic is decapsulated and re-encapsulated to get it to the final decapsulator near to the server. In principle it would be possible to tunnel direct from the remote ISP to the server subnet, but this assumes a degree of trust between ISPs that seems unlikely and unnecessary.

Traffic originating within an immediate neighbor ISP is forwarded natively to the border of the destination ISP where it is encapsulated as in the single ISP case.

# 4 Implementation Details

To achieve DoS resistance for the server-net, we need careful control over the propagation of routing information. We also need appropriate filtering capability, a filter request channel, and sufficient encapsulation capacity. We will discuss first the single ISP scenario, then explore the options for extending this to multiple ISPs, and finish by discussing feedback and encapsulation.

## 4.1 Single ISP Routing

The basic requirements of routing to implement a server-net within a single ISP are:

- Server-net addresses must only be advertised to the rest of the Internet from the encapsulators.

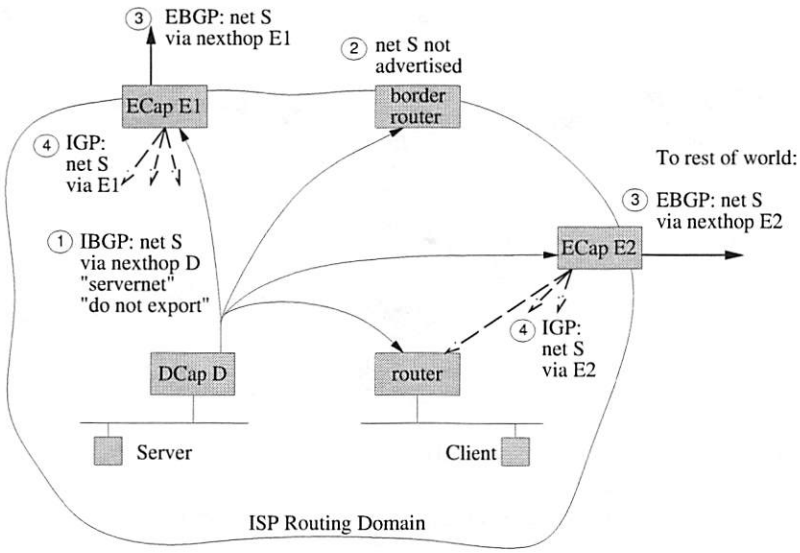



Figure 2: Route propagation for single ISP server-net

- The encapsulators need to learn which subnets are in the server-net space, and which decapsulator is associated with each.
- The addresses used by the encapsulator and decapsulator must not be advertised to the outside world.
- A server-net subnet must not be advertised to server-net hosts on other server-net subnets.

We believe these requirements can be satisfied by current BGP[9] implementations on current router hardware.

The server-net could be manually configured, but in a large ISP this will probably be unfeasible. There are many ways to do this dynamically, but one possible solution is illustrated in Figure 2 and elaborated below.

1. The decapsulator associated with a server-net subnet advertises that subnet into I-BGP[1]. It advertises its decapsulation address as the BGP nexthop router address. The route is tagged with a special BGP *server-net community*[2]. The route is also tagged with the "*do not export*" community that the ISP normally uses to indicate internal infrastructure routes that should not be exported to peers.
2. All border routers in the domain are already configured to not propagate routes to their external peers that have been tagged with the *do not export* community, so server-net routes will not leak to the outside world.
3. All encapsulators in the domain receive the server-net routes, and match on the server-net community. They then remove the *do not export* community and the *server-net community* from matching routes, and re-advertise them to external peers with the nexthop rewritten to be their own address. This will draw external traffic to the encapsulators.
4. The encapsulators also re-advertise any routes tagged with the *server-net community* into their IGP routing[3]. IGP routes are generally preferred over I-BGP routes on the basis of *administrative distance*[1], so this will cause traffic from clients within the ISP's domain to be drawn to the encapsulators rather than directly to the decapsulators.
5. The decapsulator routers also receive routes containing the *server-net community* that have been advertised by other decapsulators. The decapsulator installs a black-hole route for these subnets to prevent server-net to server-net communication.

Using routing in this way should satisfy our requirements. It is likely that slightly simpler solutions are possible if we can add BGP Path Attributes, but this is probably best done in the light of deployment experience.

## 4.2 Multiple ISP Routing

To protect its own server subnets, each ISP joining a multi-ISP server-net first runs the mechanisms described for a single-ISP server-net. To peer *within* a multi-ISP server-net, the following changes are needed:

1. Instead of an encapsulator at the border of the ISP, as shown in figure 2, a router with both encapsulation and decapsulation capability is used.
2. The encap/decap router does not remove the *server-*

[1] I-BGP: Internal BGP - using BGP to carry routing information between routers within a domain.

[2] A *community* is a locally defined BGP routing tag. The actual value of community to use can be locally decided by the ISP.

[3] IGP: Interior Gateway Protocol - the intra-domain routing within an ISP, typically OSPF or IS-IS.

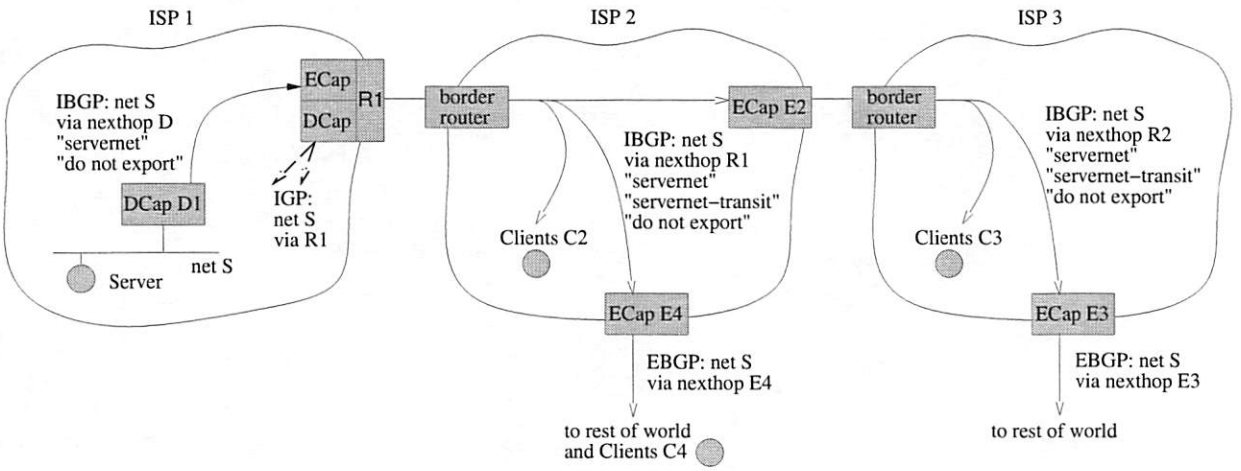



Figure 3: Route propagation for multi-ISP server-net

*net* and *do not export* communities when transmitting the route to a cooperating neighboring ISP[4].

If the number of server subnets in the multi-ISP server-net were small, then the neighboring ISP can do exactly as described in the single ISP solution. However, in a large server-net, the number of server subnets is likely to exceed the number of routes that can be safely redistributed into the IGP. In addition, even cooperating ISPs are likely to be wary of providing another ISP with a way to inject routes into their IGP. Thus we need to modify the mechanism a little, as shown in figure 3.

On receipt at the neighboring ISP, the border router will match on the *server-net* community, and add an additional *server-net-transit* community to these routes, distinguishing them from locally originated server-net routes. Routes with this additional community will *not* be redistributed into the IGP routing by encapsulators.

The result is that traffic from clients within the neighboring ISP's network (C2 in Figure 3) will reach the first encapsulator in the destination ISP using native forwarding. On the other hand, traffic from clients that would transit the neighboring ISP (C4 in Figure 3) will be encapsulated by the neighboring ISP's encapsulator and tunneled to the decapsulator at the destination ISP, before being immediately re-encapsulated and sent on to the destination server subnet's decapsulator.

Where more than two ISPs peer within a server-net, routes distributed onward to other server-net ISPs are sent with the BGP nexthop unchanged. Thus traffic will only ever be encapsulated and decapsulated twice:

- Encapsulated at the first encapsulator in the nearest server-net domain to the client.
- Decapsulated and immediately re-encapsulated at the first encapsulator in the destination ISP.
- Decapsulated at the destination subnet.

There is one issue remaining with regards to figure 3. There is no guarantee that traffic from clients C3 will traverse encapsulator E2 on its way to R1, and hence be encapsulated. To ensure this does happen requires controlling the inter-domain distribution of routes for R1. In fact the requirement is that routes for such decapsulators only transit between ISPs at the same peerings that server-net routes transit. A very similar use of an additional community tag can be made to preserve this congruence. The principal difference is that such routes are never propagated outside the server-net boundary.

## 4.3 Encapsulation and Filtering

IP-in-IP encapsulation is a standard feature on most routers. Juniper Networks ship a hardware tunnel interface module[4] capable of encapsulation at 10Gb/s that supports 8,000 tunnel virtual interfaces; other vendors no doubt have similar products. Thus current hardware is capable of performing fast tunneling to enough destinations to satisfy even large server-nets.

Most backbone routers also support packet filtering capability. It seems likely that they support sufficient filter rules to cope with attacks on the scales currently seen. In a multi-ISP server-net, any one attack is spread across many encapsulators, making it even harder for an attacker to saturate the filtering capability.

A cheaper solution is to use PC hardware. 400 Mb/s forwarding was possible in 2000[6] with minimum sized packets, limited largely by the PCI bus. A modern PC with PCI-Express is likely to be capable of in excess of 1 Gb/s with a very large number of hashed filter rules.

No standard exists for automated pushback of filters, but one would likely emerge if server-nets were widely

[4]This assumes that both ISPs use the same community values to indicate the server-net and do-not-export. If not, it will have to translate to the neighboring ISPs values.

deployed. In the meantime, a signaling channel would have to work around what is currently available.

Bro[8] can enable filter rules via the command line interface of Cisco routers. This is clunky, but works. In a similar manner, a server-net could use buddy-hosts co-located with each encapsulator. A buddy host would validate that a filter request came from the correct server-net decapsulator address by checking the BGP routing table, and then performing a handshake to prevent spoofing. It would install the filter rule in its local encapsulator using the command line interface. In the long run, we expect routers would directly support such a signaling channel.

The minimum filter granularity would likely be {source address, destination prefix} to prevent the targeting of other hosts on a victim's subnet. Also possible is {source prefix, destination prefix} to avoid an attacker spoofing multiple hosts on the same source subnet.

## 5 Future Work

Our initial plans are to work with ISPs to evaluate the feasibility of deploying a trial server-net in the wild. Based on this as well as any problems experienced, we will investigate whether it is worth standardizing additional BGP Path Attributes to simplify deployment.

A server-net neutralizes whole classes of attack, and attackers will adapt. Attacks requiring a full connection will have to become more subtle to avoid detection, and brute-force attacks will need to employ spoofing. To provide full protection, a server-net clearly needs to tackle spoofing; in the short term, servers might signal encapsulators to prioritize bidirectional flows.

In the longer term, server-net control points might be enhanced to validate source addresses. A stateful control point could perform a nonce-exchange with a source before allowing packets downstream. More specifically, in response to a data packet, a control point might send an ICMP nonce packet back to the source. The source would then need to echo the nonce in the data packet to allow the packet to pass and to instantiate state. For IPv6, the nonce could be carried in a data packet using a destination option. For IPv4, this is harder, as IP options are virtually useless. However, the IPSEC authentication header might be generalized into a form similar to the IPv6 destination option. Although such extensions are a long-term proposal, there might be incentive for deployment: a server-net under attack would give much higher priority to connection setup packets that were resent with the correct nonce echo.

## 6 Conclusions

A server-net is a protected region of the Internet that can provide upstream filtering capability for servers under DoS attack. We have sketched out how to build such a capability using current off-the-shelf router hardware and software components. The scheme we presented provides a workable solution using tunneling and careful control of routing to provide points of control in the network; filters may then be installed at these points to drop DoS traffic close to its source. Further work is needed to test real deployment of the scheme at ISPs, to understand the economics of deployment, and to limit source address spoofing.

This work is not intended to be a complete solution to DoS attacks in the Internet. It only protects servers, and only those that do not need to communicate frequently with other similarly-protected servers. In its current form, it only protects against non-spoofed attacks. However, this is still a significant improvement over unprotected directly-connected servers, especially when defending against attacks on higher-level functionality which cannot be spoofed because they require a full TCP connection to be setup. More complete solutions are likely to require significant architectural change to the Internet, and so will take very much longer to come to consensus and deploy.

## References


[1] Cisco Systems. What is administrative distance? http://www.cisco.com/warp/public/105/admin_distance.html.

[2] Communications Innovation Institute. Summary of the initial meeting of the DoS-resistant Internet working group. http://www.thecii.org/dos-resistant/meeting-1/summary.html, January 2005.

[3] Mark Handley and Adam Greenhalgh. Steps towards a DoS-resistant Internet architecture. In *Workshop on Future Directions in Network Architecture (FDNA 2004)*. ACM SIGCOMM, September 2004.

[4] Juniper Networks. Tunnel services PIC datasheet. http://www.juniper.net/products/modules/tunnel_pic.html.

[5] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of ACM SIGCOMM*, August 2002.

[6] E. Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Trans. on Computer Systems*, 18(3):263–297, August 2000.

[7] R. Moskowitz, P. Nikander, and P. Jokela. Host Identity Protocol. draft-moskowitz-hip-09.txt, work-in-progress, IETF, February 2004.

[8] Vern Paxson. Bro: A system for detecting network intruders in real-time. In *Computer Networks, 31(23-24)*, pages 2435–2463, December 1999.

[9] Y. Rekther and T. Li. A border gateway protocol (BGP-4). http://www.ietf.org/rfc/rfc1771.txt, March 1995.

[10] R. Stone. CenterTrack: An IP overlay network for tracking DoS floods. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

# Reducing Unwanted Traffic in a Backbone Network

Kuai Xu
University of Minnesota
kxu@cs.umn.edu

Zhi-Li Zhang
University of Minnesota
zhzhang@cs.umn.edu

Supratik Bhattacharyya
Sprint ATL
supratik@sprintlabs.com

## Abstract

This paper studies the techniques a backbone ISP can employ to reduce unwanted traffic on its network. For this purpose, we extract likely sources of exploit (thus unwanted) traffic from packet traces collected on backbone links using an Internet traffic behavior profiling methodology we developed earlier. We first study the characteristics of exploit traffic from several aspects, such as network origins and severity. Based on these characteristics, we propose several heuristic rules that an ISP may pursue for reducing unwanted traffic, and evaluate their cost and performance. Using packet traces collected from backbone links, we demonstrate that simple blocking strategies could potentially reduce substantial exploit traffic in a backbone network.

## 1 Introduction

Recently we have seen a tremendous increase in unwanted or exploit traffic [1] [2] – malicious or unproductive traffic that attempts to compromise vulnerable hosts, propagate malware, spread spam or deny valuable services. A significant portion of this traffic is due to self-propagating worms, viruses or other malware; this leads to a vicious cycle as new hosts are infected, generating more unwanted traffic and infecting other vulnerable hosts. In addition to self-propagating malware, new variants of old malware or new exploits emerge faster than ever, producing yet more unwanted traffic. Current measures in stopping or reducing unwanted or exploit traffic[1] rely on various firewalls or similar devices deployed on the *end hosts* or at *stub networks* (i.e., networks such as enterprise or campus networks that do not provide *transit* services) to block such traffic. In this paper we are interested in the feasibility and effectiveness of stopping or reducing unwanted traffic from the perspective of transit networks or ISPs (Internet Service Providers), in particular that of a *backbone* ISP.

As a prerequisite to stop or reduce unwanted traffic at an ISP, we first need an effective and efficient mechanism to identify such traffic and its sources, especially using packet header information of one-way traffic only. In a recent work [3], we have developed a backbone traffic profiling methodology – using a combination of information-theoretical and data mining techniques – to automatically discover and classify interesting and significant communication patterns from largely unstructured traffic data. Using packet header traces of one-way traffic collected on Sprint backbone links, we have demonstrated that our methodology is capable of identifying canonical behavior patterns for well-known servers such as the HTTP, SMTP, and DNS, as well as for traffic generated by known or unknown exploits. In addition, our methodology also uncovers "unusual" behavior patterns that deviate from the canonical profiles and thus warrant further investigation by security analysts.

Given the exploit traffic thus identified, in this paper we consider blocking strategies an ISP may pursue to reduce unwanted traffic, by installing access control lists (ACLs) on routers at entry points of an ISP. Although most of exploit traffic is associated with a relatively small set of (destination) ports, simply blocking these ports from any source is, in general, infeasible for a backbone ISP. This is because many ports that are vulnerable to attacks such as port 1434 (Microsoft SQL server) [4] or port 139 (Common Internet File System for Windows) are also used by legitimate applications run by an ISP's customers. An alternate approach is to block the specific offending sources (and the exploit destination ports) of exploit traffic. However, these sources can number in tens or hundreds of thousands for a large backbone network; hence there is a significant scalability problem (primarily due to overheads incurred in backbone routers for filtering traffic using ACLs) in attempting to block each and every one of these sources. Hence this approach is likely to be most cost-effective when used to block the top offending sources that send a majority of

self-propagating exploit traffic, in particular, in the early stage of a malware outbreak, to hinder their spread.

The contributions of this paper are i) characterizing unwanted traffic in a backbone network in terms of their sources, severity and sequential activities; ii) devising and evaluating possible blocking strategies for reducing unwanted traffic in a backbone network.

The remainder of the paper is structured as follows. In section 2 we provide a short overview of the backbone traffic behavior methodology we have developed, and apply it to identify individual sources that generate a significant amount of exploit traffic in any 5-minute time period. In section 3 we study the characteristics of extracted exploit traffic from several aspects. In section 4 we propose several heuristic blocking rules for reducing exploit traffic and evaluate their efficacy and trade-offs. In section 5 we summarize our findings and outline the future work.

## 2 Profiling Behavior of Exploit Traffic

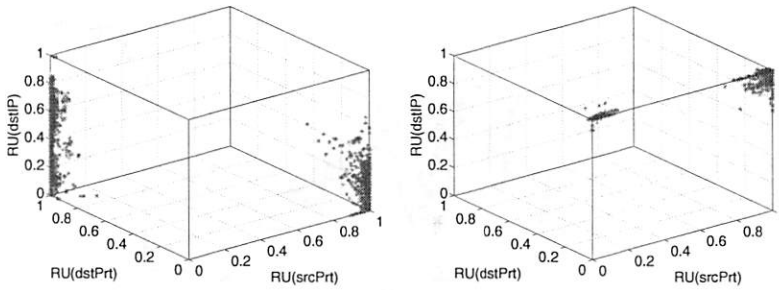
We provide a short overview of the backbone traffic behavior profiling methodology we have developed in [3]. By using a combination of information-theoretical and data mining techniques, the profiling methodology can identify several "canonical" behavior profiles such as "normal traffic" associated with typical servers and heavy-hitter client hosts, "unwanted" or exploit traffic, as well as rare or anomalous behavior patterns. The methodology is extensively evaluated and validated using packet header traces collected on backbone ISP links.

The behavior profiling works by examining communication patterns of end hosts (source and destination IP addresses) or ports (source and destination port numbers) that account for a significant number of flows in a time period (5-minute is used in this and our earlier studies). For example, for a given source IP address (`srcIP`) $a$, the profiling process includes i) extracting the 5-tuple flows whose `srcIP` is $a$ in the 5-minute time period into to a cluster, $C_a$, referred to as the *srcIP* cluster (associated with $a$); ii) characterizing the communication patterns (i.e., behavior) of $a$ using information-theoretical measures on the remaining three feature dimensions of the flows, i.e., source port (`srcPrt`), destination port (`dstPrt`) and destination IP address (`dstIP`). Note that the profiling process also works for `dstIP`, `srcPrt` or `dstPrt`.

We introduce an information-theoretic measure – *relative uncertainty*[2] ($RU_X$) – to provide an index of variety or uniformity on each of the three feature dimensions, $X$ = {`srcPrt, dstPrt, dstIP`}. Based on this measure, we define an RU vector [$RU_{srcPrt}$, $RU_{dstPrt}$ and $RU_{dstIP}$] to characterize the uncertainty of the three dimensions for each srcIP cluster. Hence each `srcIP` cluster can be represented as a single point in a 3-dimensional space of the RU vectors. This leads to a behavior classification scheme which classifies all srcIPs into various behavior classes based on their similarity/dissimilarity in the RU vector space. In particular, we identify three *canonical* behavior profiles, namely, server profile, heavy hitter profile, and exploit profile, to which most of `srcIP` clusters belong. We have applied the framework on a diverse set of backbone links and demonstrated the applicability of the profiling methodology to the problem of classifying distinct behavior patterns. For example, using the packet traces collected from an OC48 backbone link during a 24-hour period, we identified 418, 466 and 3728 distinct `srcIPs` with server, heavy hitter and exploit behavior profiles, respectively. Due to a lack of space, we will only show the results for this link, $L$, in this paper. The results for other links are presented in [5].

As an example to illustrate the distinct behaviors of *normal* vs. *exploit* traffic profiles, Figs. 1[a] and [b] plot the points in the RU vector space corresponding to the `srcIPs` belonging to the three canonical traffic profiles[3]. The points are clustered in three clearly separable groups. The points on the left side of Fig. 1[a] belong to the server profile, where they share a strong similarity in $RU_{srcPrt}$ (low uncertainty) and $RU_{dstPrt}$ (high uncertainty): a server typically talks to many clients using the same service `srcPrt` and randomly selected `dstPrt`'s. The cluster on the right side of Fig. 1[a] belong to the heavy hitter profile, where they share a strong similarity in $RU_{srcPrt}$ (high uncertainty), $RU_{dstPrt}$ (low uncertainty), and have *low-to-medium* uncertainty in $RU_{dstIP}$: a heavy-hitter client host tends to talk to a limited number of servers using randomly selected `srcPrt`'s but the same `dstPrt`. Closer inspection reveals that the `srcPrt`'s in the server profile almost exclusively are the well-known service ports (e.g., TCP port 80); whereas the majority of the `dstPrt`'s in the heavy-hitter profile are the well-known service ports, but they also include some popular peer-to-peer ports (e.g., TCP port 6346).

In contrast, the points in the exploit traffic profile (Fig. 1[b]) all have high uncertainty in $RU_{dstIP}$ and low uncertainty in $RU_{dstPrt}$, and fall into two categories in terms of $RU_{srcPrt}$. Closer inspection [4] reveals that the `dstPrts` include various known exploit ports (e.g., TCP ports 135, 137, 138, 445, UDP ports 1026-28) as well as a few high ports with unknown vulnerabilities. They also include some well-known service ports (e.g., TCP 80) as well as ICMP traffic ("port" 0). Fig. 2 plots the *popularity* of the exploit ports in $L$ in the decreasing order, where the popularity of an exploit port is measured by the number of sources that have an exploit profile associated with the port. Clearly, a large majority of these ports are associated with known vulnerabilities




(a) srcIPs with normal behavior profiles (b) srcIPs with exploit behavior profile

Figure 1: The RU vector distribution of the canonical behavior profiles for significant `srcIP`'s in $L$ during a 24-hour period.

and widely used by worms or viruses, e.g., TCP port 135 (W32/Blaster worm), TCP port 3127 (MyDoom worm). Several well-known service ports (e.g., TCP port 80, UDP port 53, TCP port 25) are also scanned/exploited by a few sources. Most sources target a single exploit, however, a small number of sources generate exploit traffic on multiple ports concurrently. In most cases, these ports are associated with the same vulnerability, for instance, the port combination {TCP port 139, TCP port 445} associated with MS Window common Internet file systems (CIFS), and {UDP ports 1026-1028} associated with MS Window messenger pop-up spams.

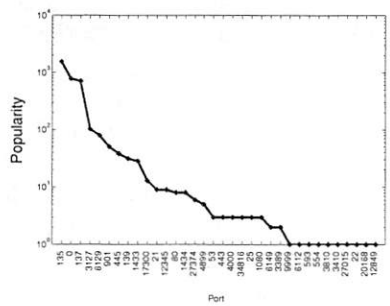



Figure 2: Port popularity of exploits traffic in $L$ during a 24-hour period

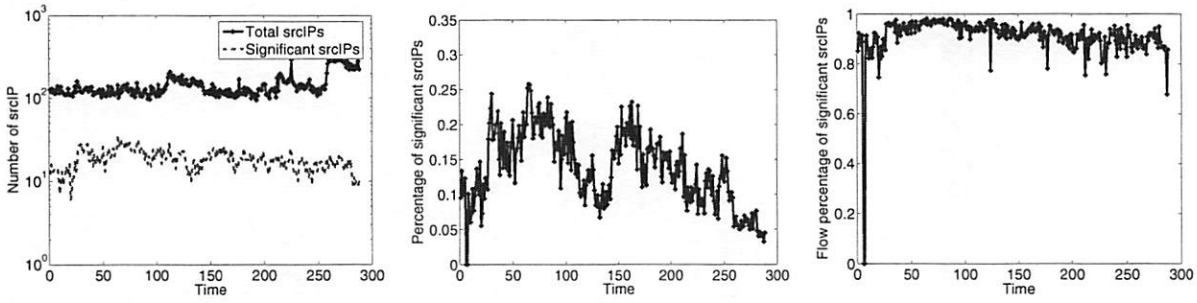
It is worth noting that our focus is on *significant* end hosts or services, so the sources we built behavior profiles are far less than the total number of sources seen in backbone links. Thus, it is not surprising that our behavior profiling framework identifies a subset of sources that send exploit traffic. However, these sources often account for a large percentage of exploit traffic. For example, Fig. 3[a] shows the total number of sources that send at least one flow on the most popular exploit port, port 135, as well as the number of significant sources extracted by our clustering technique that targeted port 135. As illustrated in Fig. 3[b], the percentage of such significant sources ranges from 0% to 26%. However, as shown in Fig. 3[c], these significant sources account for 80% traffic on TCP port 135 for most intervals. This observation suggests that our profiling framework is effective to extract most exploit traffic sent by a small number of aggressive sources.

## 3 Characteristics of Exploit Traffic

We study the characteristics of the exploit traffic from the sources profiled as exploits in section 2 in terms of network origins, their frequency, intensity and target footprints in the IP space. Our objective is to shed light on effective strategies we can explore for reducing such unwanted traffic.

### 3.1 Origins of Exploit Traffic

We first examine where the sources of exploit traffic are from, in terms of their origin ASes (autonomous systems) and geographical locations. Among the 3728 `srcIPs` in $L$ during a 24-hour period, 57 are from the private RFC1918 space [6]. These source IP addresses are likely leaked from NAT boxes or spoofed. For the remaining `srcIP`'s, we search its network prefix using the *longest prefix* match in a snapshot of the BGP routing table of the same day from Route-Views [7], and obtain the AS that originates the prefix. These 3671 `srcIP`'s are from 468 different ASes. Fig. 4 shows the distribution of the exploit sources among these ASes. The top 10 ASes account for nearly 50% of the sources, and 9 out of them are from Asia or Europe.




(a) Significant sources of exploit

(b) Percentage of significant sources

(c) Flow percentage from significant sources

Figure 3: Aggregated traffic from significant sources of exploit on TCP port 135 over a 24-hour period (i.e., 288 five-minute periods.

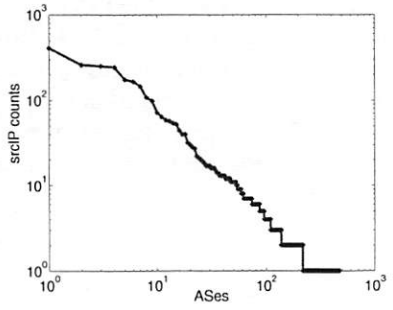



Figure 4: Distribution of srcIP counts across all ASes for 3728 sources of exploit in $L$ during a 24-hour period.

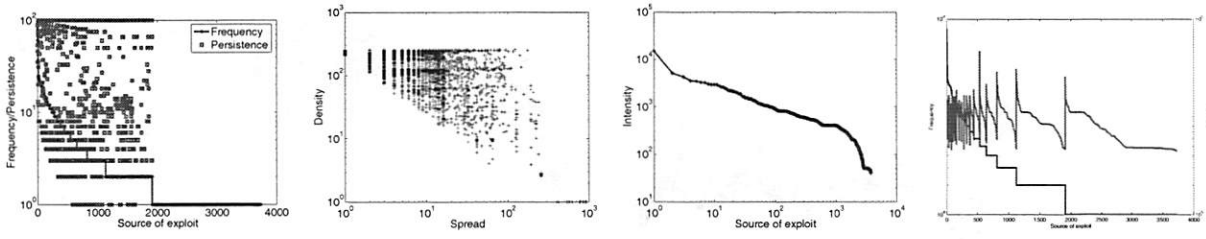
## 3.2 Severity of Exploit Traffic

We introduce several metrics to study the temporal and spatial characteristics of exploit traffic. The *frequency*, $T_f$, measures the number of 5-minute time periods (over the course of 24 hours) in which a source is profiled by our methodology as having an exploit profile. The *persistence*, $T_p$, measures (in *percentage*) the number of *consecutive* 5-minute periods over the total number of periods that a source sends significant amount of exploit traffic. It is only defined for sources with $T_f \geq 2$. Hence $T_p = 100(\%)$ means that the source continuously sends significant amount of exploit traffic in all the time slots it is observed. We use the *spread*, $F_s$, of the target footprint (i.e., destination IP address) to measure the number of $/24$ IP address blocks that a source touches in a 5-minute time period, and the *density* of the target footprint, $F_d$, to measure the (average) number of IP addresses within each $/24$ block that a source touches in the period. Finally, we use the *intensity*, $I$, to relate both the temporal and spatial aspects of exploit traffic: it measures the (average) number of distinct target IP addresses per minute that a source touches in each 5-minute period. Thus it is an indicator how fast or aggressive a source attempts to spread the exploit.

Figs. 5(a)(b)(c)(d) show the distributions of the frequency vs. persistence, a scatter plot of the spread vs. density of target footprint, the distribution of intensity, and the distributions of frequency vs. intensity for the 3728 exploit sources, respectively. From Fig. 5(a) we observe that frequency follows a power-law like distribution: only 17.2% sources have a frequency of 5 or more, while 82.8% sources have a frequency of less than 5. In particular, over 70% of them have frequency of 1 or 2. Furthermore, those 17.2% frequent ($T_f \geq 5$) sources account for 64.7%, 61.1% and 65.5% of the total flows, packets, and bytes of exploit traffic. The persistence varies for sources with similar frequency, but nearly 60% of the sources ($T_f \geq 2$) have a persistence of 100 (%): these sources continuously send exploit traffic over time and then disappear.

From Fig. 5(b) we see the exploit sources have quite diverse target footprints. In nearly 60% cases, exploit sources touch at least ten different $/24$ blocks with a density of above 20. In other words, these sources probe an average of more than 20 addresses in each block. However, in about 1.6% cases, the sources have a density of less than 5, but a spread of more than 60. In a sense, these sources are smart in selecting the targets as they have a low density in each block. As the rate of exploit seen from each destination network is slow [8], they may evade port scan detection mechanisms used, e.g., in SNORT [9], Bro [10] or [11]. Upon close examination we find that these sources employ two main strategies for target selections. One is to randomly generate targets (or to use a hit-list). The other is to choose targets like $a.b.x.d$ or $a.x.c.d$, instead of $a.b.c.x$, where $x$ ranges from 1 to 255, and $a, b, c, d$ take constant values.

The exploit intensity (Fig. 5(c)) also follows a power-law like distribution. The maximum intensity is 21K tar-




(a) Frequency ($T_f$) and persistence ($T_p$)

(b) Footprint ($F_s$ vs. $F_d$)

(c) Intensity ($I$)

(d) Frequency ($T_f$) and intensity ($I$)

Figure 5: Temporal and spatial aspects of exploit traffic for the sources with exploit profiles in the backbone link during a 24-hour period. Note that (a) and (d) have the same index in $x$ axis.

gets per minute, while the minimum is 40 targets per minute. There are only 12.9% sources with an intensity of over 500 targets per minute, while nearly 81.1% sources have an intensity of less than 500 targets per minute. Those 12.9% aggressive ($I \geq 500$) sources account for 50.5%, 53.3%, and 45.2% of the total flows, packets, and bytes of exploit traffic. However, as evident in Fig. 5(d), there is no clear correlation between frequency and intensity of exploit traffic: the intensity of exploit activities varies across sources of similar frequency.

In summary, we see that there is a relatively small number of sources that frequently, persistently or aggressively generate exploit traffic. They are candidates for blocking actions. Whereas a small percentage of sources are also quite smart in their exploit activities: they tend to come and go quickly, performing less intensive probing with wide-spread, low-density target footprint. These sources may be operated by malicious attackers as opposed to innocent hosts infected with malware that attempt to self-propagate. These sources need to be watched for more carefully.

## 4 Initial Assessment of Blocking Strategies

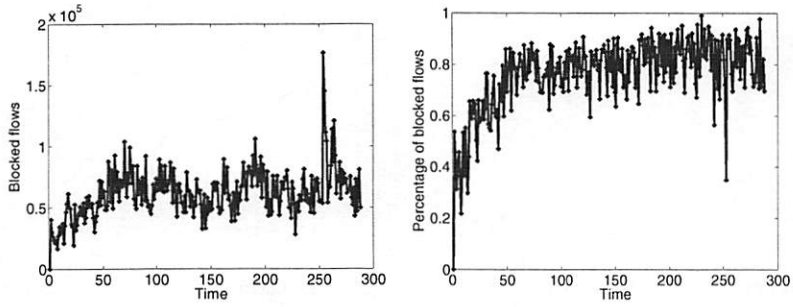
In this section, we propose several heuristic rules of blocking strategies based on characteristics of exploit activities and then evaluate their efficacy in reducing unwanted traffic.

In order to determine which sources to block traffic from, we use the behavior profiling technique outlined in section 2. For every five minute interval, we profile all sources and identify those that exhibit the exploit traffic profile. We then devise simple rules to select some or all of these sources as candidates for blocking. Instead of blocking all traffic from the selected sources, we consider blocking traffic on only the ports that a source seek to exploit. This is because exploit hosts may indeed be sending a mixture of legitimate and exploit traffic. For example, if an infected host behind a NAT box is sending exploit traffic, then we may observe a mixture of legitimate and exploit traffic coming from the single IP address corresponding to the NAT box.

For our evaluation, we start with the following benchmark rule. If a source is profiled as an exploit source during any five minute interval, then all traffic from this source on vulnerable ports is blocked from then on. Fig. 6[a][b] illustrates the total blocked flows from sources of exploit every 5-minute interval in $L$, and the percentage of such flows over all traffic from these sources, respectively. Overall, the benchmark rule could block about 80% traffic from the sources of exploit. In other words, this rule may still not block all traffic from the source due to two reasons. First, the source might already have been sending traffic, perhaps legitimate, prior to the time-slot in which it exhibited the exploit profile. Second, as explained above, only ports on which we see exploit traffic are considered to be blocked.

While this benchmark rule is very aggressive in selecting sources for blocking, the candidate set of source/port pairs to be added to the ACLs of routers may grow to be very large across all links in a network. Therefore, we consider other blocking rules that embody additional (and more restrictive) criteria that an exploit source must satisfy in order to be selected for blocking.

- *Rule 2*: an ACL entry is created if and only if the source has been profiled with an exploit behavior on a port for $n$ consecutive intervals. This rule is to block traffic from persistent sources;
- *Rule 3*: an ACL entry is created if and only if the source has an average intensity of at least $m$ flows per minute. This rule is to block aggressive sources;
- *Rule 4*: an ACL entry is created if and only if the source is exploit one of the top $k$ popular ports. This




(a) Blocked flows

(b) Percentage

Figure 6: a) blocked flows using the benchmark rule on $L$ over a 24-hour period; b) percentage of blocked flows over the total flows from sources of exploit.

rule is to block exploit traffic of the popular ports;

- *Rule 5*: Rule 2 plus Rule 3.

We introduce three metrics, *cost*, *effectiveness*, and *wastage* to evaluate the efficacy of these rules. The cost refers to the overhead incurred in a router to store and lookup the ACLs of blocked sources/ports. For simplicity, we use the total number of sources/ports as an index of the overhead for a blocking rule. The effectiveness measures the reduction of unwanted traffic in terms of flow, packet and byte counts compared with the benchmark rule. The resource wastage refers to the number of entries in ACLs that are never used after creations.

Table 1 summarizes these rules of blocking strategies and their efficacy. The benchmark rule achieves the optimal performance, but has the largest cost, i.e., 3756 blocking entries[5]. *Rule 2* with $n = 2$ obtains 60% reductions of the benchmark rule with 1585 ACL entries, while *Rule 2* with $n = 3$ obtains less than 40% reductions with 671 entries. *Rule 3*, with $m = 100$ or $m = 300$ achieves more than 70% reductions with 2636 or 1789 entries. *Rule 4* has a similar performance as the benchmark rule, but its cost is also very high. The *Rule 5*, a combination of *Rule 2* and *Rule 3* has a small cost, but obtains about 40% reductions compared with the benchmark rule.

We observe that the simple rules, *Rule 3* with $m = 100$ or $m = 300$ and *Rule 2* with $n = 2$, are most cost-effective when used to block the aggressive or frequent sources that send a majority of self-propagating exploit traffic, in particular, in the early stage of a malware outbreak, to hinder their spread.

## 5 Conclusions and Ongoing Work

This paper studied the characteristics of exploit traffic using packet-level traffic traces collected from backbone links. Based on the insights obtained, we then investigated possible countermeasure strategies that a backbone ISP may pursue for reducing unwanted traffic. We proposed several heuristic rules for blocking most offending sources of exploit traffic and evaluated their efficacy and performance trade-offs in reducing unwanted traffic. Our results demonstrate that blocking the most offending sources is reasonably cost-effective, and can potentially stop self-propagating malware in their early stage of outburst. We are currently performing more in-depth analysis of exploit traffic, and correlating exploit activities from multiple links. Ultimately we plan to incorporate these mechanisms in a comprehensive security monitoring and defense system for backbone ISPs.

## Acknowledgments

Kuai Xu and Zhi-Li Zhang were supported in part by the National Science Foundation under the grants ITR-0085824 and CNS 0435444 as well as ARDA grant AR/F30602-03-C-0243. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

We thank Travis Dawson at Sprint ATL for many helpful comments and discussions.

## References

[1] V. Yegneswaran , P. Barford and J. Ullrich, "Internet intrusions: global characteristics and prevalence," in *Proc. of ACM SIGMETRICS*, 2003.

Table 1: Simple blocking strategies and their efficacy.

| Rule | Cost | Effectiveness (Reduction (%)) | | | Wastage |
|---|---|---|---|---|---|
| | | flow | packet | byte | |
| Benchmark | 3756 | - | - | - | 1310 |
| Rule 2 (n=2) | 1586 | 63.0% | 61.2% | 56.5% | 505 |
| (n=3) | 671 | 38.0% | 36.0% | 31.2% | 176 |
| Rule 3 (m=100) | 2636 | 97.1% | 94.0% | 89.4% | 560 |
| (m=300) | 1789 | 84.3% | 80.4% | 72.7% | 302 |
| (m=500) | 720 | 57.6% | 57.0% | 53.1% | 68 |
| Rule 4 (k=5) | 3471 | 87.4% | 79.2% | 77.5% | 1216 |
| (k=10) | 3624 | 92.9% | 85.5% | 81.5% | 1260 |
| Rule 5 (n=2, m=300) | 884 | 48.7% | 44.0% | 37.7% | 163 |


[2] R. Pang, V. Yegneswaran, P. Barford, V. Paxson and L. Peterson, "Characteristics of Internet Background Radiation," in *Proc. of ACM SIGCOMM Internet Measurement Conference*, 2004.

[3] K. Xu, Z.-L. Zhang and S. Bhattacharyya, "Profiling Internet Backbone Traffic: Behavior Models and Applications," in *Proc. of ACM SIGCOMM*, August 2005.

[4] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford and N. Weaver, "Inside the Slammer Worm," *IEEE Security and Privacy*, July 2003.

[5] K. Xu, Z.-L. Zhang and S. Bhattacharyya, "Reducing Unwanted Traffic in a Backbone Network," Sprint ATL Research Report RR05-ATL-040400, April 2005.

[6] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, "RFC1918: Address Allocation for Private Internets," February 1996.

[7] University of Oregon, "Routeviews archive project," http://archive.routeviews.org/.

[8] S. Staniford, J. Hoagland, and J. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, pp. 105–136, 2002.

[9] "SNORT," http://www.snort.org/.

[10] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, Dec 1999.

[11] J. Jung and V. Paxson and A. Berger and H. Balakrishna, "Fast portscan detection using sequential hypothesis testing," in *Proc. of IEEE Symposium on Security and Privacy*, 2004.


## Notes

[1] Strictly speaking, in this paper we will use the term *exploit* traffic to mean traffic that is generated with the explicit intention to exploit certain vulnerabilities in target systems - a large subset of *unwanted* traffic, although frequently we do use the two terms interchangeably.

[2] Suppose the size of $C_a$ is $m$ and $X$ may take $N_X$ discrete values. Moreover, $P(X)$ denotes a probability distribution, and $p(x_i) = m_i/m, x_i \in X$, where $m_i$ is the frequency or number of times we observe $X$ taking the value $x_i$. Then, the RU of X for $C_a$ is defined as $RU(X) := \frac{H(X)}{H_{max}(X)} = H(X)/\log\min\{N_X, m\}$, where $H(X)$ is the (empirical) *entropy* of $X$ defined as $H(X) := -\sum_{x_i \in X} p(x_i) \log p(x_i)$.

[3] For clarity of presentation, points belonging to the *rare* behavior classes, i.e., those falling outside the three canonical behavior profiles, are excluded in both plots. These rare behavior classes tend to also contain anomalous or suspicious activities. See [3] for more details.

[4] Our profiling approach reveals the dominant activity of a given source, and not all activities. For example, an infect host, which sends a large number of exploit traffic, could also send legitimate web traffic.

[5] The cost exceeds the total number of unique sources of exploit since a few sources have exploit profiles on multiple destination ports.

# Analyzing Cooperative Containment of Fast Scanning Worms

Jayanthkumar Kannan, Lakshminarayanan Subramanian, Ion Stoica, and Randy H. Katz
*Computer Science Division, University of California, Berkeley*
*{kjk, lakme, istoica, randy}@cs.berkeley.edu*

## Abstract

Fast scanning worms, that can infect nearly the entire vulnerable population in order of minutes, are among the most serious threats to the Internet today. In this work, we investigate the efficacy of cooperation among Internet firewalls in containing such worms. We first propose a model for firewall-level cooperation and then study the containment in our model of cooperation using analysis and simulation. Our results suggest that, with moderate overhead, cooperation among Internet firewalls can provide 95% containment under 10% deployment while being resilient to 100-1000 malicious firewalls.

## 1 Introduction

Scanning worms that probe IP addresses to find vulnerable hosts are the most common worms today. Several recent worms, such as Slammer, Witty, CodeRed, and Blaster, fall in this category. Slammer, one of the fastest scanning worms seen so far, took less than 10 minutes to infect 90% of the Internet's vulnerable population [1]. Scanning worms that are more carefully designed using techniques discussed in [2] can accomplish infection even faster. This threat is further acerbated by the possibility of worms that exploit unknown vulnerabilities. It is clear that automatic mechanisms are required to defend against these worms. Unfortunately, many existing defenses, like patching and address blacklisting, hold out little hope in containing novel fast worms [3].

An emerging approach that offers hope in containing such worms is a decentralized solution in which firewalls in various access networks exchange information amongst themselves to defend against worm attacks. Such a scheme may be easier to deploy compared to end-host based schemes [4–6], schemes that may require support from core routers [7, 8], or schemes that place the burden of analysis on a centralized infrastructure [9, 10]. Recent work [11–14] suggests that such *cooperation* may contain fast worms, but an understanding of its efficacy and limitations on robustness is still lacking.

In this paper, we aim to fill this void by modeling cooperation in a abstract fashion, and then studying the containment in this model. In our model, the participants in the cooperative are Internet firewalls, each protecting an access network. Each firewall employs *local detection* to detect infection within its network. Once it detects an infection, it notifies other firewalls of the ongoing worm infection. We consider two forms of such notifications: *implicit* and *explicit* signaling. Firewalls that receive these signals are alerted to the worm attack and can then *filter* their incoming traffic based on these signals. Signaling between firewalls is the essence of cooperation, and our model of signaling captures and extends the salient features of existing schemes for cooperation. Moreover, this model integrates various known techniques for local detection [11, 15, 16] and filtering [4, 5, 17] into a framework for cooperation.

Our analytical results based on this model are as follows. First, we show that local detection and filtering without any signaling can contain scanning worms under certain regimes. Second, we characterize the dependence of the containment provided by cooperation on various worm and cooperation parameters. Our results indicate that the containment offered by cooperation drops linearly with the scan rate of the worm and the average time taken by the firewall to detect infection, and drops quadratically with the size of the vulnerable population. Third, our analysis quantifies the trade-off between the containment offered by cooperation and robustness to malicious participants. Our numerical results suggest that, even against the most virulent worm seen so far, cooperation among Internet firewalls can offer over 95% containment under 10% deployment while being resilient to about 100-1000 malicious firewalls.

Although we primarily study Internet-level cooperation among firewalls to contain scanning worms, our analytical approach can be applied to other scenarios as well:

to enterprise-level or Internet-level cooperation, to host-level or firewall-level cooperation, and to a limited extent for analyzing sophisticated worms like hit-list based worms. The downside of our analytical approach however is that there are inherent limitations in any tractable model amenable to analysis. Such a model, by necessity, cannot include all "real-life" details. Our model includes the main factors characterizing a worm attack and we use simulations to incorporate the effect of other factors in our results.

## 2 Modeling Cooperation

We consider the Internet to consist of $N$ access networks that are connected to their ISPs through an access link. Each network is monitored by a firewall which analyzes traffic on its access link and exchanges information with other firewalls to contain worms. We focus on containment of fast random scanning worms in this work. Such worms perform a local topological scan followed by a global uniform random scan: an infected host first infects all hosts in its network and then probes external IP addresses uniformly at random to find more vulnerable hosts. Note that, in our model, a firewall cannot prevent infection from spreading within its network, since it may not even see local topological scans.

The main goal of cooperation is to maximize the containment metric $C$, defined as the fraction of vulnerable networks that escape infection during a worm attack. Other desirable features of a cooperative containment scheme include resistance to malicious participants, effectiveness under partial deployment, and scalability (*e.g.,* bandwidth overhead). We now describe our model of cooperation.

### 2.1 Framework for Cooperation

The main purpose of this framework is to define the design space of cooperation schemes in a simple manner suitable for analysis. Some points in this space correspond to existing schemes [11–13, 18, 19].

We begin by making two assumptions, which we will remove later: (a) all firewalls operate according to our protocol (b) all firewalls in the Internet participate in our cooperative. Firewalls participating in the cooperative perform three functions: local detection, propagation, and filtering.

**Local Detection:** A firewall uses local detection to determine whether its own network is infected by analyzing *outgoing* traffic. There are two advantages in analyzing outgoing traffic as opposed to analyzing incoming traffic. A firewall can maintain characteristics of its outgoing traffic using per-host state, unlike incoming traffic which can be noisy due to background Internet traffic. Also, a decision made using incoming traffic can potentially be influenced by external malicious hosts sending traffic to the firewall's network. However, analyzing outgoing traffic cannot aid a firewall in protecting its *own* network, since the characteristics of outgoing traffic change only *after* its network is infected. We use the term "detected firewall" to denote a firewall whose local detection scheme identifies infection within its network.

One can use existing techniques [5, 11, 15, 20] for local detection. These techniques identify infection by analyzing various characteristics of outgoing traffic, *e.g.,* number of unique destination addresses, connection failure rate, and packet payload.

**Propagation:** A detected firewall proceeds to notify other firewalls of its infection. These notifications include *filters* for identifying malicious packets. On receiving these notifications, these firewalls are said to be "alerted" to the attack. There are two forms of such notifications: *implicit* and *explicit*.

*Implicit Signaling*: A detected firewall sends implicit signals by *marking* all suspicious outgoing packets. Suspicious packets are identified using filters inferred by the filtering mechanism. Marking can be done by using bits in packet headers or by encapsulating suspicious packets within special headers. This marking serves two purposes. First, it informs the destination firewall that the packet is possibly malicious. The destination firewall can simply drop such packets or it can process them in a special manner (*e.g.,* send it to a hardened end-host for analysis). Thus, the decision to drop/process is delegated to the destination. Second, the destination firewall is notified that the source network is infected. This enables the destination firewall to install filters before its own network gets infected: this is the essence of cooperation.

*Explicit Signaling*: A detected firewall sends explicit notifications of its infection to other participating firewalls at some fixed rate $E$. We only consider schemes where signals are sent to randomly chosen participating firewalls. This naive method can easily be improved by, say, a publish-subscribe based system; however the security properties of such systems are harder to characterize.

In both types of signaling, note that a firewall can report only of its *own* infection; it cannot implicate other firewalls. This rules out *setup* attacks in which malicious firewalls make false accusations. We use a challenge-response protocol to verify that the originator of a signal is the infected firewall itself. Such a protocol also eliminates spoofing attacks by malicious end-hosts.

**Filtering:** We refer to a firewall that has been alerted by a signal as an *alerted* firewall. Such a firewall installs filters and drops malicious *incoming* traffic matching these filters. These filters are gathered from the sig-

nals it receives. Filtering is also used by a detected firewall during implicit signaling: only outgoing packets identified as malicious by the filter are marked.

A very simple filter could be based on port numbers: any traffic on alerted ports is dropped. More sophisticated filters can be inferred using known techniques, such as Autograph [17] which deduces signatures from packet payloads, and Vigilante [5], TaintCheck [4], which infer signatures by host-based mechanisms to track the flow of network data in a program.

To summarize, in our scheme, every firewall is in one of the following four states: normal, alerted, infected, and detected. Our model can also be extended to the case when multiple worms may be propagating. The filtering mechanism would infer filters for each propagating worm, and firewalls can maintain state and perform propagation on a per-worm basis.

## 2.2 Security

We now discuss the security of our model against malicious firewalls and evasive worms. First, in a cooperative framework, it is important to distinguish between a global Internet-wide attack and an infection localized to a small set of networks. Thus, if a small number of firewalls report that they have been infected, even if they are being truthful, this attack could simply be a localized one. Thus, a fundamental necessity in the cooperative framework is that a firewall should enter the alerted stage and install filters only if it receives signals from $T$ distinct firewalls. Second, it is also necessary to defend against firewalls attempting to trigger false positives/negatives. Since the alerted state is entered only upon receiving $T$ signals, our scheme can resist up to $T$ malicious firewalls that trigger a false alarm. This design choice is also fundamental since it may be impossible to detect whether a notification sent by a firewall is a false alarm (*e.g.,* if a firewall originates traffic as if its own network had been infected). False negatives occur when a firewall deliberately/otherwise fails to report of its infection. We have also devised schemes to defend against such attacks [21], although we do not discuss them here.

Finally, cooperation should also be resilient to sophisticated worms that attempt to evade propagation (worms that evade local detection and filtering are beyond our scope). A worm could use pre-generated hit-lists [2] to ensure that most of its probes are successful, and also alter its scanning pattern over time to thwart propagation. In the limit, a worm could simply stop scanning after its firewall has entered the detected stage, since beyond this point, all its scans will be marked and cannot infect any new hosts. In this case, implicit signaling is completely ineffective. Explicit signaling will however continue to perform well. We consider such a worm in Section 4.2.

## 2.3 Partial Deployment

Under partial deployment, the main limitation of our model is that it is not possible to prevent undeployed networks from getting infected eventually, since infected hosts in undeployed networks can always infect other undeployed networks. This limitation cannot be overcome without support from core routers. We therefore choose our metric for the partial deployment case as the containment among the *deployed* firewalls. As a baseline solution, our existing scheme for propagation works in this case as well. Only deployed firewalls perform detection and propagation: undeployed firewalls do not engage in local detection, propagation, or filtering. We also improved on this scheme using a technique called rerouting discussed in [21]; we omit details due to lack of space.

## 2.4 Discussion

There are three salient features in our model of cooperation. First, in our model, signals can be *verified* using a challenge-response mechanism. This makes the security properties of our model much easier to characterize as compared to existing work [13, 14]. Second, our model incorporates both forms of signaling, implicit and explicit, whereas existing work has dealt mainly with explicit signaling. There are trade-offs associated with both approaches; implicit signaling has lesser overhead and may be simpler to implement, but explicit signaling is necessary to contain smart worms. Finally, our model integrates the various techniques proposed for local detection and filtering in a framework for cooperation. Decoupling the various mechanisms used in cooperative containment also aids analysis, as will be seen in the following section.

# 3 Analyzing Cooperation

In this section, we analyze the containment metric $C$ (the fraction of vulnerable networks that escape infection) of our cooperative model under three cases: all firewalls are deployed and operate according to the protocol (Section 3.2, 3.3,3.4), all firewalls are deployed but some of them may be malicious (Section 3.5), and finally under partial deployment (Section 3.6).

## 3.1 Worm and Network Model

We now introduce the assumptions and the parameters used in our analysis (parameter names are in bold type). In our worm model, an infected host first infects all vulnerable hosts in its own network in *zero* time, and then

proceeds to probe external addresses. The scanning rate of a fast worm is limited by access bandwidth, and hence we use the scanning rate **"s"** of a single *infected network* in our analysis. The probability of a successful probe **"p"** is the number of vulnerable hosts divided by the size of the IP address space. We denote the total number of vulnerable networks as **"N"**. We use the homogeneous cluster model, used for modeling Slammer victims in [1]. This assumes that the vulnerable hosts are uniformly distributed among all vulnerable networks.

For purposes of analysis, we assume that the time taken for an infected firewall to detect its infection is an exponential variable with mean **"$t_d$"**. This assumption is certainly not true for all local detection schemes, but considerably simplifies the analysis. Thus, our model includes several simplifications, such as the uniform distribution of vulnerable hosts. In the following results, the environment used for simulation exactly matches the one used in analysis, and thus, the simulation only serves as a validation for the analysis. We also simulated environments with non-uniform host distributions and network delays, and obtained similar results (not presented here for lack of space).

## 3.2 Effectiveness of Detection and Filtering

First, we analyze a simplified variant of our scheme without any propagation under complete deployment. Thus, there is no sharing of information between firewalls. In this variant, a firewall performs only local detection and filtering (once a firewall enters the detected stage, outgoing scans are marked and cannot infect any more hosts). Thus, the defense against a worm attack is that an infected host can effectively probe only until its firewall does not detect infection. The following lemma shows that such a simplified scheme is surprisingly effective under some conditions. We define $\lambda$ as the expected value of the number of successful infections by an infected network before its firewall detects infection. $\lambda$ can be written in terms of our parameters as $\lambda = spt_d$. We now state the following lemma without proof (for all proofs, see our technical report [21]).

**Lemma 1** *If $(\lambda < 1)$, then as $N \to \infty$, the containment metric $C \to 1$. Further, if $I_0$ denotes the number of infected firewalls at time $t = 0$, for any $N$, $C \geq 1 - \frac{I_0}{(1-\lambda)N}$.*

Interestingly, the above lemma holds irrespective of the scanning strategy employed by the worm. The condition $\lambda < 1$ can be enforced by deploying firewalls at as fine a granularity as necessary. This can be used to control the effective scanning rate $s$ and thus the parameter $\lambda$. Earlier worms like Blaster had low scan rates leading to low values of $\lambda$, and could have been controlled by simple detection and filtering even if the firewalls were deployed at the level of class $B$ networks. Recent worms like Slammer used much higher scan rates and require very fine grain deployment to enforce $\lambda < 1$.

Surprisingly, even if $\lambda > 1$, detection and filtering still provide some containment against a *random scanning* worm, although the effectiveness degrades rapidly with $\lambda$. The reason this occurs is that as the infection proceeds, it takes longer for a random scanning worm to find uninfected hosts, and thus the local detection schemes have a greater chance of throttling the infection.

**Lemma 2** *If $(\lambda > 1)$, assuming $I_0 \ll N$, $C \geq 1 - \min_{k:k>1}(\frac{(k\lambda-1)(k+1)}{k\lambda(k-1)} - \frac{2*\log(k\lambda)}{(k-1)\lambda})$ against a random scanning worm ($k$ is a variational parameter used in minimization).*

Our analytical results based on the above lemma indicate that detection and filtering provide about $40\%$ containment at $\lambda = 1.5$ and about $20\%$ at $\lambda = 2.0$. We show later that $\lambda = 2.0$ corresponds to the one of the most virulent worms seen so far.

## 3.3 Effectiveness of Implicit Signaling

In this section, we analyze the containment metric when implicit signaling is used along with detection and filtering under complete deployment. Denote the total number of deployed firewalls as $n$. We model the infection process as follows. At time $t$, denote by $n_i(t)$ the number of infected firewalls, by $n_d(t)$ the number of detected firewalls, and by $n_a(t)$ the number of alerted firewalls (the remaining firewalls are in the "normal" state). Note that $n_i(t)$ represents all infected firewalls, so $n_i(t) - n_d(t)$ is the number of infected firewalls that have not yet detected infection. A simple differential equation model for implicit signaling is as follows:

$$\frac{d}{dt}(n_i) = (n_i - n_d) \times s \times \frac{(n - n_i - n_a)}{N} \quad (1)$$

$$\frac{d}{dt}(n_d) = \frac{n_i - n_d}{t_d} \quad (2)$$

$$\frac{d}{dt}(n_a) = n_d \times s \times \frac{(n - n_i - n_a)}{N} \quad (3)$$

The first and third equations count the number of successful scans and alerts per unit time respectively. The second equation is based on the assumption of an exponential distribution for detection: the probability that an infected and undetected network identifies infection during time $dt$ is equal to $dt/t_d$. We could not solve these equations exactly, so we present a closed-form upper bound:

**Lemma 3** *For $\lambda > 1$ and $I_0 \ll N$, with implicit signaling, $C \geq 1 - \frac{(log(N))t_d\sigma^2)}{s}(\frac{1}{t_d\sigma}+1)$ where $\sigma = (\lambda-1)/t_d$ and $\lambda = pst_d$.*

By substituting for $\sigma$ in the region $\lambda > 1$, the above lemma suggests that the containment metric decreases linearly with $t_d$ and $s$ and drops quadratically with $p$.

### 3.4 Effectiveness of Explicit Signaling

In the case of explicit signaling at rate $E$ along with implicit signaling, the containment metric can be obtained by simply substituting $s = s + E$ in the denominator of Lemma 3 to obtain $C \geq 1 - \frac{log(N)t_d\sigma^2}{s+E}(\frac{1}{t_d\sigma} + 1)$. Notice that the containment metric varies with the rate of explicit signaling as $1/(1 + E/s)$.

### 3.5 Effectiveness under Threshold $T$

In this case, a firewall enters the alerted state after receiving alerts from $T$ distinct firewalls. Our differential equation model in Section 3.3 can be extended to model this case by adding $(T + 1)$ differential equations to track the number of firewalls that have received $0, 1, \ldots, (T-1), T$ alerts. This approach is however cumbersome even for low values of $T$. However, the lower bound on containment metric for the implicit signaling case (Lemma 3) can be applied to obtain:

**Lemma 4** *For $\lambda > 1$ and $I_0 \ll N$, the containment metric $C$ obtained by implicit signaling is at least $1 - \frac{(log(N)+(T-1)log(log(N)))t_d\sigma^2)}{(s+E)}(\frac{1}{t_d\sigma} + 1)$ where $\sigma = \frac{\lambda-1}{t_d}$.*

The main implication of this result is that the containment drops linearly with the threshold $T$. This quantifies the trade-off between the robustness of cooperation and the containment $C$.

### 3.6 Effectiveness under Partial Deployment

The differential equation model in Section 3.3 can be suitably modified by introducing a differential equation for the number of infected undeployed firewalls at time $t$. Unfortunately, we could not obtain a closed-form analytical bound (as in the complete deployment case). We numerically solve our differential equation model to evaluate cooperation under partial deployment.

## 4 Numerical Solutions

We now evaluate the containment offered by cooperation against worms of varying virulence and how this containment depends on the various mechanisms used in cooperation. Our results are obtained by two means: (a) numerical integration of the differential equation model (b) discrete-time event simulation to validate the analytical method (marked "Sim" in our plots). The number of initial infected networks is set to 10 by default, and the simulation is run until all the networks are infected/alerted.

### 4.1 Default Parameters

In the following results, the default parameters are as follows. We used $p = 0.0005$ corresponding to a vulnerable population of about 2 million, twice that of Blaster, one of the most wide-spread worms. We set the scanning rate $s$ as 20000 (the average scan rate of Slammer [1], one of the fastest worms so far). Thus, these settings correspond to one of the most virulent worms seen yet. The number of firewalls $N$ was set to 100000 based on the number of observed BGP prefixes (obtained from routeviews.org). Thus, every network has an address space of $2^{15}$ and about 20 vulnerable hosts. Under these settings, the worm takes 0.6 seconds to infect 99% of all vulnerable hosts, assuming no network congestion. We set $t_d = 0.2s$ in our analysis and use implicit signaling for propagation. By default, we assume all networks are deployed. Note that $\lambda = 2$ under these settings, so as per Lemma 2, signaling is necessary for containment under these settings.

### 4.2 Varying Worm Virulence

We present results corresponding to three axes of worm virulence in order to evaluate the performance of cooperation against known worms and worms of the future.

**Number of Vulnerable Hosts:** The size of the vulnerable population is a key parameter of worm virulence, and we plotted the containment metric against this parameter in Figure 1(a). The number of vulnerable networks was kept constant, and the vulnerable population size was varied between 2 and 20 million hosts (corresponding to $p = 0.0005 - 0.005$). As suggested by the analysis, there is roughly a quadratic drop in $C$ with $p$.

**Scanning Rate:** Figure 1(b) plots the containment metric against the worm scanning rate, since the propagation time of the worm is also influenced by this rate. The containment metric exhibits a slow linear drop with scanning rate of the worm. Since implicit signals are piggybacked on worm scans, faster the worm scanning rate, greater the effective signaling rate as well.

**Number of Initial Seeds:** Our analytical bounds assume that the set of hosts used to seed the infection is a small subset of the vulnerable population. This assumption may not be true if botnets (networks consisting of already infected zombie hosts) are used to seed a worm attack, given that botnets of up to 50000 hosts have been discovered. At one extreme, if all these hosts are in different networks, our results indicate 48.63% containment. At the other extreme, either all hosts within a net-

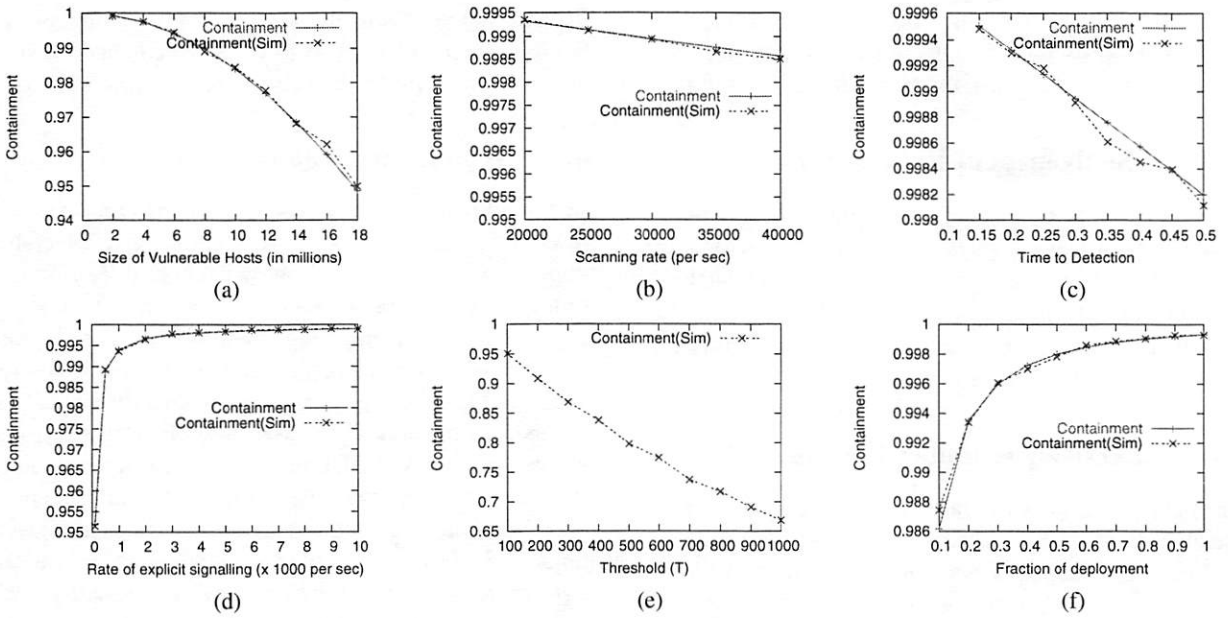



Figure 1: (a) Containment Vs. Size of vulnerable population (b) Containment Vs. Scanning Rate (c) Containment Vs. Time to detection (d) Containment Vs. Rate of Explicit Signaling (e) Containment Vs. Threshold $T$ (f) Containment Vs. Deployment

work belong to the botnet or none of them do. In our settings, each network has 20 vulnerable hosts, which means that 2500 networks are under attacker control. In this case, cooperation provides 96.88% containment.

## 4.3 Varying Cooperation Parameters

This section presents results illustrating the impact of detection time $t_d$, signaling parameters (explicit signaling rate $E$ and threshold $T$), and level of deployment, on containment.

**Local Detection:** Figure 1(c) illustrates the sensitivity of cooperation to local detection. As suggested by the analysis in the previous section, the dependence is roughly linear. Though the worm propagates in about $0.6s$, even with time to detection $t_d$ as high as $0.5s$, cooperation performs very well. This might appear surprising, but note that $0.5s$ seconds is the *mean* time to detection, and with a finite probability, detection occurs before $0.5s$.

**Rate of Explicit Signaling:** The rate of explicit signaling corresponds to the overhead of cooperation, and we plotted containment metric obtained through explicit signaling for varying rates $E$ (100-10000) in Figure 1(d). In obtaining this plot, only explicit signaling is used. The containment metric improves with the ratio $E$ although the variation is very low.

**Threshold $T$:** Since the security of the scheme is determined by the threshold $T$ (number of alerts required to transition to the alerted state), in Figure 1(e), we varied the threshold $T$ from 100 to 1000 firewalls (0.1% to 1.0% of the total number of firewalls). Since the differential equation model is cumbersome for threshold $T \gg 1$, we show only the results obtained through simulation. The containment drops by as much as 30%, and clearly $T$ is the most sensitive of all the parameters we have considered so far. This means that in our model, cooperation is resilient to a small fraction of malicious firewalls, but cannot deal with large scale collusion (for example, when the worm infects firewalls).

**Level of Deployment:** Figure 1(f) shows the containment obtained under various levels of deployment. Cooperation continues to perform well even at deployment levels as low as 10%. The reason is that as soon as a certain fraction of deployed firewalls are infected, the propagation of alerts outpaces the worm scan rate.

# 5 Related Work

We classify related work into three main categories. The first includes works that analyze traffic at an end-host or a single observation point to detect worm attacks, *e.g.,* Vigilante [5], Throttling [15], TaintCheck [4], Threshold random walks [16, 20], Honeypot-based architectures. Any of these proposals can serve as a local detection scheme in our model (in a firewall or host-based cooperative). In general, host-based mechanisms can detect a wider class of worms, but involve a heavy deployment cost. The second category relies on inferring and deploying filters to contain worms *e.g.,* Autograph [17], Dynamic Quarantine [7]. Filter placement within the Internet core is necessary for most of these schemes. In

contrast, existing mechanisms for inferring filters can be used as filtering mechanisms in our model to achieve good containment without modifying core routers. The final category relates to proposed architectures for worm containment involving multiple vantage points. Architectures, such as [9], perform distributed data collection followed by centralized analysis. Decentralized designs include Hard Perimeters [11], Weaver et al [12], Domino [18]. In [11,12], data collection and analysis are performed by firewalls, while [18] splits these functionalities between satellites and an axis overlay. All participants are implicitly trusted in these proposals. Our model accomodates malicious participants by using simple verification along with a thresholding mechanism. Nojiri et al [13] handles malicious participants, but unlike our model, assumes the existence of apriori trust relationships among firewalls. Anagnostakis et al [14] proposes a signaling protocol very similar to our explicit signaling, but their focus is more on multiple propagating worms as against fast worms. Finally, Staniford et al [22] proposes an analytical model for worm containment, which however does not study cooperative containment.

## 6 Conclusion

In this work, we have discussed a framework for modeling cooperation in order to conduct a preliminary analysis of its efficacy, resilience and deployability. This framework is intended to be as general as possible, and to capture and extend the notion of cooperation proposed in recent work. The advantage of analyzing a general framework is that our results can be applied to cooperation in a wide-variety of scenarios: at the enterprise-level or Internet-level, at the host-level or firewall-level, and also to a limited extent for analyzing sophisticated worms like hit-list based worms. Our analysis can be used to decide if cooperation is necessary, and if so, can help tune local detection and signaling.

In an Internet-level cooperative of firewalls, our results indicate that cooperation can indeed be an effective solution to containing fast uniform scanning worms, and may also be effective against more sophisticated worms of the future. Cooperation can be made resilient to a small number of malicious participants (100-1000), and even at low levels of deployment (about 10%), effective containment can be achieved. As part of future work, we are working on hybrid protocols that use implicit signaling in the early stages of a worm, and switch to explicit signaling beyond a threshold.

## References


[1] N. Weaver, I. Hamadeh, G. Kesidis, and V. Paxson, "Preliminary results using scaledown to explore worm dynamics," in *Proc. ACM CCS WORM*, Oct 2004.

[2] Stuart Staniford, Vern Paxson, and Nicholas Weaver, "How to Own the Internet in Your Spare Time," in *Proc. USENIX Security*, Aug 2002.

[3] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage, "Internet quarantine: Requirements for containing self-propagating code," in *Proc. INFOCOM*, Apr 2003.

[4] James Newsome and Dawn Song, "Dynamic taint analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," in *Proc. NDSS*, Feb 2005.

[5] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Colleen Shannon, and Jeffery Brown, "Can we contain Internet worms?," in *Proc. HOTNETS*, Nov 2004.

[6] Christina Warrender, Stephanie Forrest, and Barak A. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *IEEE Symposium on Security and Privacy*, May 1999.

[7] Cynthia Wong, Chenxi Wang, Dawn Song, Stan Bielski, and Gregory R. Ganger, "Dynamic Quarantine of Internet Worms," in *Proc. DSN*, Jun 2004.

[8] V. Berk, G. Bakos, and R. Morris, "Designing a framework for active worm detection on global networks," in *IEEE Intl Workshop on Information Assurance*, Mar 2003.

[9] Nicholas Weaver, Vern Paxson, and Stuart Staniford, "Wormholes and a Honeyfarm: Automatically Detecting Novel Worms," in *DIMACS Large Scale Attacks Workshop*, Sep 2003.

[10] J. Wu, S. Vangala, L. Gao, , and K. Kwiat, "An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques," in *Proc. NDSS*, Feb 2004.

[11] Nicholas Weaver, Dan Ellis, Stuart Staniford, and Vern Paxson, "Worms vs Perimeters: The Case for HardLANs," in *Proc. Hot Interconnects*, Aug 2004.

[12] Nicholas Weaver, Stuart Staniford, and Vern Paxson, "Very Fast Containment of Scanning Worms," in *Proc. USENIX Security*, Aug 2004.

[13] D. Nojiri, J. Rowe, and K. Levitt, "Cooperative Response Strategies for Large Scale Attack Mitigation," in *Proc. DISCEX*, Apr 2003.

[14] K. G. Anagnostakis, M. B. Greenwald, S. Ioannidis, A. D. Keromytis, and D. Li, "A Cooperative Immunization System for an Untrusting Internet," in *Proc. ICON*, Oct 2003.

[15] J. Twycross and M. M. Williamson, "Implementing and testing a virus throttle," in *Proc. USENIX Security*, Aug 2003.

[16] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Proc. IEEE Security and Privacy*, May 2004.

[17] H. A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proc. Usenix Security*, Aug 2004.

[18] Vinod Yegneswaran, Paul Barford, and Somesh Jha, "Global Intrusion Detection in the Domino Overlay System," in *Proc. NDSS*, Feb 2004.

[19] C. G. Senthilkumar and K. Levitt, "Hierarchically Controlled Co-operative Response Strategies for Internet Scale Attacks," in *Proc. DSN*, June 2003.

[20] Stuart E. Schechter, Jaeyeon Jung, and Arthur W. Berger, "Very Fast Containment of Scanning Worms," in *Proc. RAID*, Sep 2004.

[21] Jayanthkumar Kannan, Lakshminarayanan Subramanian, Ion Stoica, Scott Shenker, and Randy Katz, "Cooperative containment of fast scanning worms," Tech. Rep. CSD-04-1359, UC Berkeley, Nov 2004.

[22] S. Staniford, "Containment of scanning worms in enterprise networks," *Journal of Computer Security*, 2005 (to appear).

# Push vs. Pull: Implications of Protocol Design on Controlling Unwanted Traffic

Zhenhai Duan
*Florida State University*

Kartik Gopalan
*Florida State University*

Yingfei Dong
*University of Hawaii*

## Abstract

In this paper we argue that the difficulties in controlling unwanted Internet traffic, such as email SPAM, stem from the fact that many Internet applications are fundamentally *sender-driven* and distinctly lack *receiver control* over traffic delivery. However, since only receivers know what they want to receive, receiver-driven approaches may often have clear advantages in restraining unwanted traffic. In this paper, we re-examine the implications of the two common traffic delivery models: *sender-push* and *receiver-pull*. In the sender-push model, a sender can deliver traffic at will to a receiver, who can only passively accept the traffic, such as in the SMTP-based email delivery system. In contrast, in the receiver-pull model, receivers can regulate *if and when* they wish to retrieve data, such as the HTTP-based web access system. We argue that the problem of unwanted Internet traffic can be mitigated to a great extent if the receiver-pull model is employed by Internet applications, whenever appropriate. Using three popular applications – email, mobile text messages, and asynchronous voice messages – as examples, we demonstrate that asynchronous communication protocols can be easily designed using the receiver-pull communication model to suppress unwanted Internet traffic.

## 1 Introduction

In recent years the Internet has been increasingly plagued by the seemingly-never-ending unwanted traffic, manifesting itself in large volumes of unsolicited bulk emails (spam), frequent outbreaks of virus/worm attacks, and large scale Distributed Denial of Services (DDoS) attacks. For example, it was estimated that 32 billion spam messages were sent daily on the Internet as of November 2003 [11]. Worse, spammers and virus/worm attackers are increasingly joining force to automate spamming by hijacking (home) user machines through virus/worm attacks. A recent study reported that as high as 80% of spam messages were sent from compromised user machines (zombies) [15]. In this paper, we focus our attention on *spam-like* unwanted Internet traffic, which plagues critical Internet applications and services such as emails, mobile text messages, and asynchronous voice messages (where a recorded voice message is sent to a list of receivers). We refer to such applications collectively as *message services*. In this paper, we are especially interested in the implications of the protocol design on controlling unwanted traffic on the Internet.

Given the importance of controlling spam for preserving the value of the messaging systems, this issue has attracted a great amount of attention in both networking research and industrial communities. Many different spam control schemes (in the context of Internet emails) have been proposed, and some of them have been deployed on the Internet [3, 8, 9, 12, 13, 14]. On the other hand, despite these anti-spam research and development efforts, the proportion of spam seen on the Internet has been continuously on the rise. It is estimated that nowadays spam messages constitute 79% of all business emails, up from 68% since the US federal Can-Spam Act of 2003 took effect in January 2004 [2]. It was also reported that 80% of mobile phone text messages were unsolicited in Japan [16], where SMS (Short Message Services) is popular, and is therefore attractive to spammers.

In this paper we argue that the difficulties in restraining spam can be attributed to the lack of *receiver control* over how messages should be delivered on the Internet. For example, in the current SMTP-based email delivery architecture [10], any user can send an email to another at will, regardless of whether or not the receiver is willing to accept the message. In the early days of the Internet development, this was not a big problem as people on the network largely trusted each other. However, since the commercialization of the Internet in mid-1990, the nature of the Internet community has changed. It has become less trustworthy, and email spam is possibly one of

the most notable examples of the untrustworthy nature of the Internet.

In order to effectively address the issue of spam in the untrustworthy Internet, we argue that *receivers must gain greater control over if and when a message should be delivered to them.* Asynchronous messages on the Internet are delivered primarily using two different models: sender-push and receiver-pull (or a combination of the two). They differ in who initiates the message delivery process. In the sender-push model, senders control the delivery of traffic, and receivers passively accept whatever the senders push to them. The current SMTP-based email delivery system is a typical example of this model. In contrast, the receiver-pull model grants receivers the control over if and when they want to retrieve data from the senders. In this model, senders can only prepare the data but they cannot push the data to receivers. Examples of the receiver-pull model include the HTTP-based web access services and the FTP-based file transfers.

As we will discuss in the next section, the receiver-pull model comes with several appealing advantages because it grants receivers greater control over the message delivery mechanism. It takes advantage of the fact that receivers have more reliable knowledge of what traffic they want to receive. Moreover, the receiver-pull model may also simplify the challenging issues related to the resource usage accountability and sender authentication. For example, because spammers need to store and manage email messages on their own mail servers (waiting for receivers to pull), it becomes relatively easier to hold spammers responsible for the resources they consume. As a proof of concept, in this paper we present examples of three asynchronous messaging applications – emails, mobile text messages, and asynchronous voice messages.

The objective of the paper is two-fold. First, through the example designs of the message applications, we would like to demonstrate the feasibility and advantages of using receiver-pull model to design protocols for asynchronous messaging applications. Second, and more importantly, we want to raise the explicit awareness of the difference between the sender-push and receiver-pull models, and argue that, the receiver-pull model should be the strongly favored design choice, whenever appropriate.

The rest of the paper is structured as follows. In Section 2 we elaborate on the two different traffic models on the Internet. We outline the example design to support emails, mobile text messages, and asynchronous voice messages using the receiver-pull model in Section 3. We summarize the paper in Section 4.

## 2 Push vs. Pull: Implications of Protocol Design Choice

The choices made during protocol design phase have fundamental implications on security, usability, and robustness of any distributed message delivery system. One such important design decision is whether to adopt a sender-push or a receiver-pull model or a combination of the two models (see Figure 1). In this section we discuss the implication of these design choices and make the case that the receiver-pull model can prove to be highly effective in discouraging unwanted traffic.

### 2.1 The Sender-Push Model

In the sender-push model, the sender knows the identity of a receiver in advance and pushes the message in an asynchronous manner to the receiver. The receiver accepts the entire message, may choose to optionally examine the message, and then accept or discard it. An important aspect of sender-push model is that the entire message is received before any receiver-side processing is performed. A number of communication services in the Internet rely on the sender-push model. A prime example is email in which the sender relies on the Simple Mail Transfer Protocol (SMTP) to push an entire email message to a passive receiver. Asynchronous voice messages over the telephone network (both traditional and IP based) represent another important application of the sender-push model.

A common variant of the sender-push concept is the *receiver-intent-based sender-push* (RISP) model. The most common examples of the RISP model are the subscription-based services such as mailing lists, where user subscribes to a service which subsequently pushes the data to the receiver. Other popular subscription-based applications of the RISP model include stock and news ticker applications and automatic software updates. Similarly, Instant Messaging is another application where the message itself is pushed by the sender, but the receiver can allow or disallow messages from specific users.

A common feature among all the above examples is that the content itself is pushed to the receiver, whereas the receiver may optionally provide minimal control feedback to the sender. The primary advantage of the sender-push model is that its asynchronous message delivery framework is conceptually simple and fits naturally for many useful applications such as email, text, and voice messaging. Sender initiates message transfer when the message is ready, the receiver simply waits passively for any message to arrive and accepts one when it does arrive. Furthermore, there is no significant storage requirement on the sender side.

The biggest disadvantage of the sender-push model is

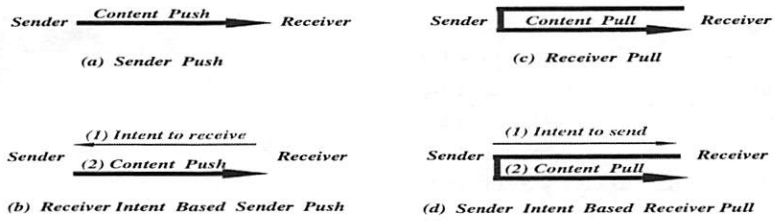



Figure 1: Common message delivery models.

that it is the sender who completely controls *what* message is delivered and *when* it is delivered. The receiver has neither the knowledge of what message he/she will receive, nor when the message will be received. The receiver is ideally expected to receive the entire message before processing or discarding it. Apart from generating and transmitting the message, the sender does not commit any resources for the transmitted message. On the other hand, the receiver has to wait, receive, process and store (or discard) the message even if the message is not of interest to the receiver.

The RISP model alleviates this concern to some extent by allowing receivers to provide control feedback. However it is not easy to implement in many popular applications. For example, adopting the RISP model for email, mobile text and voice messages requires the receiver to maintain an exhaustive white-list or black-list of email addresses and phone numbers of potential senders. Indeed, approaches such as Reverse Black Lists (RBL) [13] adopt this philosophy in trying to blacklist email spammers. However most potential correspondents, such as first time senders, fall in neither of the two categories. To handle such unclassified cases, receivers end up relying on content-based-filters, i.e. they receive the entire message, scan it to determine if it is wanted and then either accept or discard it. *The fundamental problem here lies in having to accept and examine the entire message before culling it.*

An additional disadvantage of the sender-push model is that the sender can vanish (go offline) immediately after pushing unwanted content to the receiver. This makes it quick and easy for a malicious sender to hide its identity. Once the receiver accepts the content, it is difficult at best to trace back a malicious sender.

In summary, while the sender-push model is both simple and convenient, it comes with a serious baggage, namely, that senders control what to send and when to send, and cannot be easily held accountable for sending unwanted content to receivers.

## 2.2 The Receiver-Pull Model

In the receiver-pull model, it is the receiver who initiates the message transfer by explicitly contacting the sender. The sender passively waits for the receiver and delivers the entire content upon receiving a request. Since it is the receiver who initiates the message transfer, the receiver would have explicit greater control over the message transfer and implicit greater trust in the received content, than in the sender-push model.

A number of successful communication services rely on the receiver-pull model. The most important examples using the receiver-pull model are the FTP and HTTP protocols. In both cases, the receiver initiates the data transfer by opening an FTP connection or by typing/clicking on a URL, respectively. (Interestingly, HTTP supports both receiver-pull and as well as RISP variant of sender-push, though the former is more commonly used. Examples of RISP model techniques in HTTP include automatic page refreshes and the hugely unpopular popup windows).

An interesting and useful variation of receiver-pull model, which is of special interest to us, is the *sender-intent-based-receiver-pull* (SIRP). In this model, the sender first expresses an intent to send content to the receiver via a small intention message. If the receiver happens to be interested, it contacts the sender and retrieves the content. A common example of the SIRP model is the *pager* service. Here the caller expresses an intent to talk to a callee by paging the latter and leaving a callback number. If the callee is interested, he/she contacts the caller back on the callback number. The main feature of the SIRP model is that the content itself is pulled by the receiver whereas only a short intent is pushed by the sender.

The primary advantage of the receiver-pull model is that a receiver exercises control over when and what it receives. The receiver has the freedom to first determine its own level of interest in the content (as well as the reputation of the sender) *before* it actually requests the content. Secondly, it becomes the responsibility of the sender to store and manage the content till the receiver is ready to retrieve it. For instance, an FTP or web server needs to store and manage its own files whereas receivers access it only when they are interested. Thirdly, there is a large window of time over which a malicious sender is forced to stay online and reveal its identity. For the pure receiver-pull model, this window is from the mo-

ment content is generated and named till the content is retrieved by the receiver. For the SIRP model, this window is from the moment sender expresses its intent to send till the time receiver retrieves the content. Thus, unlike the sender push model, there is a large window of time in which the receiver is free to verify a sender's identity.

One obvious disadvantage of receiver-pull model is that the sender is burdened with greater content management complexity. The sender needs to store outgoing messages and keep them available at least till the intended receivers are willing to retrieve them, and needs to have a deletion policy if a message is never retrieved by the receiver. Another issue that the sender needs to grapple with is to ensure that the party retrieving a message is indeed the originally intended receiver. However, another angle to look at these disadvantages is that, in the sender-push model, it is the receiver who needs to deal with the very same issues.

## 2.3 Implications on Unwanted Traffic

Given that the receiver-pull model grants more control to receivers in terms of traffic delivery, and only receivers know what they want to receive, the receiver-pull model has clear advantages in restraining unwanted traffic compared to the sender-push model. Moreover, the above discussion also makes it clear that the sender is accountable to a greater degree in the receiver-pull model than in the sender-push model. This brings us to the following key idea which underlies the theme of this paper: *When designing any communication protocol, it is advantageous to first consider using a receiver-pull model which inherently provides greater protection against unwanted traffic.*

The receiver-pull based model is a relatively low-cost design choice that can be considered early during any communication system design. Even if the receiver-pull model results in slightly greater protocol complexity, it can greatly help to simplify accountability and authentication issues by placing the overheads where they truly belong – at the sender of the unwanted traffic.

A legitimate concern with a receiver-pull model is that it may end up increasing the cost of sending messages for malicious as well as legitimate senders. We will show in the next section through an example of a receiver-pull based email architecture that, using simple design optimizations, one can easily lower the sending cost for legitimate senders while still holding senders of unwanted content accountable.

We do not claim that a receiver-pull based model may be universally suitable for all forms of communications. For example, soldiers in the middle of a desert war may not want to rely on remote senders being reachable when trying to retrieve their messages. However, in many important applications, such as civilian use of email, mobile text messages, and asynchronous voice messages, the receiver-pull architecture appears to offer strong advantages in fight against unwanted traffic.

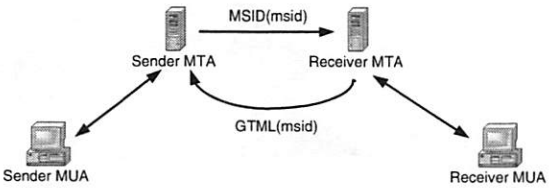



Figure 2: An email delivery architecture with receiver-pull model.

# 3 Applications of the Receiver-Pull Model

In order to illustrate the feasibility and advantages of the sender-intent-based-receiver-pull (SIRP) model in supporting asynchronous applications, in this section we outline the design of three important applications using the model: emails, mobile text messages, and asynchronous voice messages. We present the design of the SIRP based email system in greater detail and briefly sketch the design for the other two applications using a framework similar to the email design. (IM2000 [1] is another email architecture using the receiver-pull model, however it is not backward compatible.) We emphasize that these designs only illustrate the feasibility and effectiveness of supporting message services using the SIRP model, in reducing unwanted traffic. Many design details are omitted (see [4, 5] for supporting the Internet email application using the SIRP model).

## 3.1 SIRP based email System

In the SIRP based email delivery system, senders cannot directly push messages to arbitrary receivers. Instead, receivers decide if and when they want to retrieve (or pull) messages from senders. Figure 2 illustrates the basic architecture of the new email delivery system. In the following we will present the new system from both the senders' and receivers' perspectives. Before we delve into details, it is worth noting that the new system extends the current Simple Mail Transfer Protocol (SMTP) [10] by adding two new commands: MSID and GTML. In other words, all the commands and reply codes in SMTP are also supported in the new system. We will explain the two new commands when we use them.

### 3.1.1 Sender: Message Composition and Receiver Notification

Like in the current email architecture, a sender uses a Mail User Agent (MUA) to compose outgoing messages [10]. After a message is composed by the sender, the sender delivers the message to the sender Mail Transfer Agent (MTA). For simplicity, we refer to a sender MTA server as an SMTA, and a receiver MTA server as an RMTA.

All the outgoing messages are stored at the SMTA. For this purpose, the SMTA maintains an outgoing message folder for each *sender*. Instead of the complete message being directly pushed from the SMTA to the RMTA, only the envelopes (headers) of the messages are delivered. In particular, the SMTA notifies the RMTA about a new message by the new *message identifier* command MSID, which contains the unique identifier *msid* of the message. The identifier of a message is generated based on the sender, the message, the receiver, and a secret key of the sender.

We note that there is a fundamental difference between message pull in the new email delivery system and URL embedded in many current spam messages. The address in the URL is normally not related to the sending machine of the message, which makes it hard to identify the actual sender who is responsible for the spam message. On the other hand, outgoing messages in the new email system have to be stored on the sender mail servers instead of third-party machines before they are retrieved. In this way, we obtain several advantages in restricting spam. For example, senders need to keep their mail servers up until the messages are retrieved by receivers. This presents less flexibility for senders to move around by frequently changing their IP addresses and/or domains. In contrast, in the current (sender-push) SMTP-based architecture, spammers can send a large number of spam messages and shut down their mail servers, which makes it hard to hold spammers responsible for spamming. Moreover, in the new system, senders have greater responsibility to store and manage their outgoing email messages in comparison to the current email architecture, which imposes negligible responsibility on the senders. In summary, while the current SMTP-based email delivery architecture provides a *call-by-copy* interface to senders, the new system provides a *call-by-reference* interface to senders [6].

### 3.1.2 Receiver: Pulling Messages from Senders

The new email delivery system grants more control to receivers regarding if and when receivers want to read a message, senders cannot arbitrarily push a message to them. Receivers can be discriminate about which messages need to be retrieved, and which ones need not. If the receiver indeed wants to read a message, he will inform his own RMTA, and the RMTA will retrieve the message from the SMTA on behalf of the receiver. An RMTA retrieves an email message using a *get mail* command **GTML**, which includes the identifier *msid* of the message to be retrieved. After the message has been pulled to the RMTA, conventional virus/worm scanning tools and content-based spam filters can be applied to further alert the receiver about potential virus or spam. Therefore, *the new email delivery system does not exclude the use of existing email protection schemes*. For security reasons, when an SMTA receives the **GTML** command, it needs to verify that the corresponding message is for the intended receiver, and more importantly, the requesting MTA is the mail server responsible for the receiver (i.e. the one which was originally contacted for message delivery).

By only delivering the envelope (including *msid*) of a message from a sender to the receiver, less bandwidth, storage, and processing time is used at the receiver side, which is especially important for resource constrained users, e.g., wireless, PDA, or dial-up users. On the other hand, if the receiver indeed wants to read the message, negligible extra time and bandwidth is required. Since the receiver is less likely interested in messages from unknown sources, the majority of such messages will not be retrieved. As a result, considering the huge volume of spam on the Internet, much less bandwidth will be wasted by spam. For simple back of envelope calculation, assuming there are 30 billion spam messages sent daily on the Internet [11] and the average size of these messages is 5 KBytes [7]. We further assume the envelope of these messages occupies 1KBytes on average. Then it is easy to see that we will have daily 120 Tera Bytes worth of bandwidth saving on the Internet. Note that if content-based filter is used alone, these spam messages are still delivered on the Internet.

### 3.1.3 Differentiating Message Deliveries

The simple SIRP model not only puts more burden on spammers but also regular contacts of a receiver. To address this issue a hybrid email delivery system can be designed to support both the sender-push and receiver-pull models. In such a system, each receiver maintains a list of regular contacts, whose complete messages can be directly pushed from the senders to the receiver using the current SMTP protocol. In addition, a list of blacklisted contacts can be summarily declined. Messages from non-regular contacts should be stored and managed by the sender mail servers, and only the envelopes of such messages are directly delivered to the receiver to notify the pending messages.

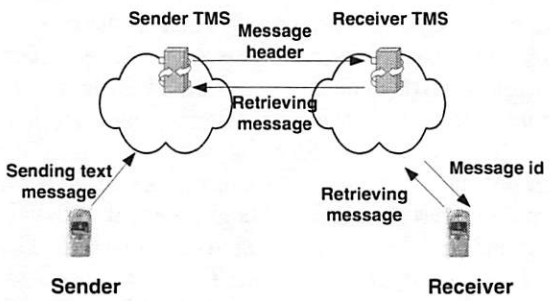



Figure 3: Supporting mobile text messages with SIRP model.

#### 3.1.4 Practical Deployment Considerations

It can be shown that the new email delivery system can be deployed incrementally, and popular message applications such as mailing lists can also be supported [4, 5].

### 3.2 Mobile Text Messages and Asynchronous Voice Messages

Figure 3 illustrates the architecture in supporting mobile text messages using the SIRP model. Each mobile phone service provider will deploy one or multiple text message servers (TMS). When a user sends a text message to another user (who may be with another provider), the text message is stored in the sender provider's TMS, and only the message header (including the corresponding phone number and a message id) is sent to the receiver provider's TMS. The receiver provider's TMS will notify the receiver about the message header. If the receiver wants to read the message, the receiver provider's TMS will retrieve the message from the sender provider's TMS on behalf of the receiver.

Asynchronous voice messages are currently supported by cell phone service providers, where a recorded voice message is sent to a receiver, or a group of receivers. This service can be potentially exploited by spammers given its capability to send a voice message to a large number of receivers with relatively little effort. Moreover, as the service is being integrated into VoIP based applications, it becomes even more attractive to spammers. This service can be supported using the SIRP model instead of the sender-push model essentially in the same manner as mobile text messages. We skip the detailed discussion due to space considerations.

## 4 Summary

In this paper we examined the fundamental implications of the two different traffic delivery models, sender-push vs. receiver-pull, on controlling unwanted traffic on the Internet. Using examples of three popular applications – email, mobile text messaging, and asynchronous voice messaging – we illustrated that the receiver-pull model can be effectively used for asynchronous messaging in place of the current sender-push model to reduce unwanted Internet traffic. Another important contribution of this paper is that, by examining the implications of two traffic delivery models, we attempt to raise explicit awareness of the impact of the two models on unwanted Internet traffic, and argue that, a receiver-pull model should be strongly favored, whenever appropriate.

## References

[1] BERNSTEIN, D. Internet Mail 2000 (IM2000). *http://cr.yp.to/im2000.html.*

[2] CLABURN, T. Big guns aim at spam. *Information Week* (Mar. 2004).

[3] DELANY, M. Domain-based email authentication using public-keys avertised in the DNS (domainkeys). Internet Draft (Aug. 2004). [draft–delany-domainkeys–base–01.txt].

[4] DUAN, Z., DONG, Y., AND GOPALAN, K. DiffMail: A differentiated message delivery architecture to control spam. Tech. Rep. TR-041025, Department of Computer Science, Florida State University (Oct. 2004).

[5] DUAN, Z., GOPALAN, K., AND DONG, Y. Receiver-driven extensions to SMTP. Internet Draft (May 2005). [draft–duan–smtp–receiver–driven–00.txt].

[6] FU, K. Personal communication. MIT, (Mar. 2005).

[7] GOMES, L., CAZITA, C., ALMEIDA, J., ALMEIDA, V., AND MEIRA, W. Charactering a spam traffic. In *Proceedings of IMC'04* (Oct. 2004).

[8] GRAHAM, P. A plan for spam . *http://www.paulgraham.com/spam.html* (2003).

[9] JUELS, A., AND BRAINARD, J. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of NDSS-1999 (Networks and Distributed Security Systems)* (Feb. 1999).

[10] KLENSIN, J. Simple mail transfer protocol. RFC 2821 (Apr. 2001).

[11] LAURIE, B., AND CLAYTON, R. "Proof-of-Work" proves not to work. http://www.apache-ssl.org/proofwork.pdf (May 2004).

[12] LYON, J., AND WONG, M. Sender ID: Authenticating e-mail. Internet Draft (Aug. 2004). [draft–ietf–marid–core–03.txt].

[13] RBL. Real-time spam black lists (RBL). http://www.email-policy.com/Spam-black-lists.htm.

[14] RISHI, V. Free lunch ends: e-mail to go paid. *The Economic Times* (Feb. 2004).

[15] SANDVINE INCORPORATED. Trend analysis: Spam trojans and their impact on broadband service providers (June 2004).

[16] THE WASHINGTON POST. FCC sets sights on mobile phone spam (Mar. 2004).

# Detecting Spam in VoIP Networks

Ram Dantu, Prakash Kolan
*Dept. of Computer Science and Engineering*
*University of North Texas, Denton*
*{rdantu, prk0002}@cs.unt.edu*

## Abstract

Voice over IP (VoIP) is a key enabling technology for the migration of circuit-switched PSTN architectures to packet-based networks. The problem of spam in VoIP networks has to be solved in real time compared to e-mail systems. Many of the techniques devised for e-mail spam detection rely upon content analysis and in the case of VoIP it is too late to analyze the media after picking up the receiver. So we need to stop the spam calls before the telephone rings. From our observation, when it comes to receiving or rejecting a voice call people use social meaning of trust and reputation of the calling party. In this paper, we describe a multi-stage spam filter based on trust, and reputation for detecting the spam. In particular we used closed loop feedback between different stages in deciding if the incoming call is a spam or not. For verifying the concepts, we used a laboratory setup of several thousand soft-phones and a commercial grade proxy server. We verified our filtering mechanisms by simulating the spam calls and measured the accuracy of the filter. Results show that multistage feedback loop fares better than any single stage. Also, the larger the network size, the harder to detect a spam call. Further work includes understanding the behavior of different controlling parameters in trust and reputation calculations and deriving meaningful relationships between them.

## 1. Introduction

Defending the country's telecommunication networks requires cooperation between service providers, equipment vendors, enterprises and the government. Currently VoIP infrastructure is being aggressively deployed in enterprises and residential areas without much security analysis. It is estimated that by 2006 IPPBX deployments will outnumber the traditional PBX deployments. This can be a clear recipe for a possible disaster to critical infrastructure like telecommunications network. There is very little work reported in the literature on how to defend VoIP against attacks like DOS (Denial of Service), session hijacking and termination, monitoring and eavesdropping, service disruption, toll fraud, identity fraud, spamming etc.. Also, the impact of vulnerabilities on a large scale (e.g., several millions of IP phones) VoIP network is not well understood. Hence it is imperative that we investigate the vulnerabilities and threats to residential communities due to the new real-time services like VoIP. All the threats need to be addressed before VoIP services are deployed on a mass scale because the lack of security has the potential of delaying and disrupting next generation voice communications.

The possibility of VoIP network replacing the PSTN network depends on enhancing the existing IP network to carry voice traffic. With the usage of IP network to carry voice traffic, existing problems on the IP network holds for the VoIP network too. One of the major issues that the present day IP networks face is the problem of controlling spam - the unsolicited bulk mail. Spam control has been perceived to be the most important problem of research with present traditional e-mail systems. The problem of spam is increasing day-by-day and recent results indicate that of all the e-mail that is circulating in the internet right now, as high as 80% of that is spam (junk or unsolicited messages). A study[12] by Radicati Group a California based consultancy states that "last year daily global e-mail traffic via the Internet amounted to 56.7 billion messages per day. Of that, the firm says, 25.5 billion messages were spam, or about 45%. Daily traffic is expected to rise above 68 billion messages per day, and more than half of it--52%--will be spam. With this magnitude of junk or spam messages circulating all through the internet every day, the problems like low availability, network congestion etc. would not be a surprise. In VoIP networks, spam refers to the unsolicited voice calls, which end up consuming many resources on the end VoIP phones and intermediate VoIP infrastructure components. With the advent of VoIP and openness of internet, the spamming attacks on the VoIP infrastructure are estimated to take the world to the same position as the traditional e-mail systems with respect to e-mail spam. While there are many techniques that have been designed to avoid e-mail spam, such techniques can be of limited application to avoid the problem of voice spam. The reason lies in the real time application of VoIP. *The problem of spam in VoIP networks has to be solved in real time compared to e-mail systems. Compare receiving an e-mail spam at 2:00 AM that sits in the Inbox until you open it next day morning to receiving a junk voice call at the same time. Moreover, many of the techniques devised for e-mail*

*spam detection rely upon content analysis. The same with VoIP calls is already late.*

## 2. Background

Most of the present day e-mail spam filters employ content filtering as both the signaling and media arrives at the spam filter at the same time. Content filtering is not useful in VoIP spam analysis as media flows in after the two participating entities have agreed upon to start the communication and would be too late to filter the call. This poses a serious challenge of detecting spam in real time with the available signaling messages and a danger of increasing the end-to-end delay on the communication between participating entities during call set up.

There is a lot of literature on spam filtering for the present day e-mail infrastructure. Spam filters have known to use a wide variety of filtering mechanisms like text classification and rule based scoring systems, Bayesian filtering, pattern recognition, identity recognition etc [1][2][3][6][7][8][10]. Cohen[8] recommends spam filtering based on a set of rules for identifying the message body content. Features of the message are identified and scored to compute the total spam score of the e-mail spam message and the messages having a score more than a given threshold is identified to be spam e-mail. Large quantities of spam and legitimate messages are used to determine the appropriate scores for each of the rules in the rule-based scoring systems. Sakkis[7] suggests probabilistic inference for calculating the mutual information index (MI) and a vector of attributes having the highest MI scores is constructed for spam identification. The Memory-based algorithms attempt to classify messages by finding similar previously received messages by storing all training instances in a memory structure, and using them directly for classification. Soonthornphisaj[6] spam filtering technique works by constructing the centroid vector of the e-mail and is classified based on its similarity measured between the centroid vector of spam e-mail class and the legitimate e-mail class. Rigoutsos[10] suggests pattern discovery scheme for identifying unsolicited e-mails by training the system with a large number of spam messages. The system matches the e-mail message with the available patterns; more the patterns are matched more is likelihood that the message is spam. Sahami[1] proposes that incorporating domain specific features in addition to identifying various textual phrases and probabilistically inferring the spam behavior of the constructed message vector leads to a more accurate spam analysis. All the solutions account for some sort of identification and filtering based on message body content. These solutions do not have direct applicability to VoIP systems, as content filtering cannot be achieved before the users communicate.

The standard for VoIP, SIP (Session Initiation Protocol), establishes an open model where users have IP phones linked to the pervasive Internet infrastructure. To realize the objective of receiving a call from a person anywhere in the world, static junk call filtering mechanisms have to be replaced with adaptive learning systems. These systems apart from learning spam behavior have to account for modeling human behavior. For example, whenever a phone rings, we first look into our state of mind (or presence), and see if the call is from a trusted party. If we do not know who the caller is, then we guess the trust and reputation of the calling party. After picking up the telephone, we query and move forward only when we are satisfied with the response. Similarly, our proposed research uses an intelligent call admission control consists of the presence of the called party (e.g., state of mind, location), the rate of incoming calls from a given user (by computing first and second order differentials), trust between calling and called parties (using Bayesian theory), and reputation graphs based on the social network of the calling party. In addition, all the above techniques are combined in deciding whether to accept/reject a call or forward it to voice mail. We propose a *Voice Spam Detector(VSD)* acting as a separate process running along with the domain proxy and processes the incoming call and informs the proxy about the spam nature of the call based on past feedback from the end users in its domain.

## 3. Methodology

VoIP spam detection process does not pertain to a single technique of detection. The detection needs to be done using various techniques at different stages. At each stage the spam detection process qualified by that stage eliminates most of the spam and any subsequent spam left through or forwarded would be quarantined in the next stage. The techniques employed at each stage would determine the spam behavior of the call and with the available feedback information from the called domain end user, the call is either stopped or forwarded to the user voicemail box. The basic criterion on which the call processing depends is on whether a similar call had been designated as a spam or a valid call before.

### 3.1 Architecture

The architecture behind the spam detection process would take into account all the user preferences of wanted and unwanted people, his or her presence of mind, the reputation and trust of the calling party. The

basic architecture would be as shown in Figure. Each stage represents a technique based on which the call would be quarantined by employing a specific set of mechanisms and user feedback. Each stage of the spam detection process gives feedback about the possibility of the call to be spam and the collective inference of all stages would give the spam nature of the call that can be used for quarantining the call.

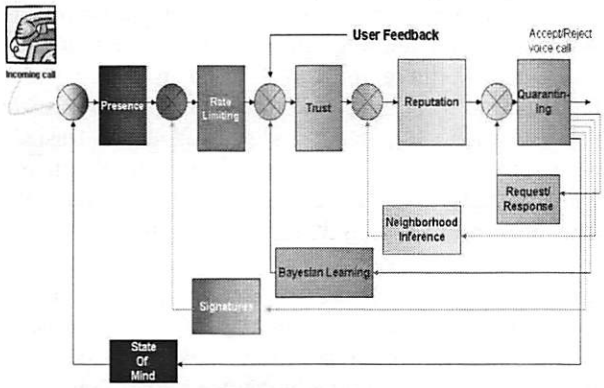



**Fig 1:** Functional Elements of VSD

### 3.2 Functional Elements in Voice spam Detection

**Presence:** Whenever we receive a voice call, we normally pick up the telephone receiver depending on our state of mind. So, the definition of a spam call depends on one's 'state of mind'. Hence the first step in this filtering process is the characterization of spam depending on the state of mind. For example, a state of mind changes depending on a location, do-not-disturb-me mode, follow-me mode, and 911-emergency-mode. One example of assessing the state of mind is to synchronize the system with an individual's calendar. The filtering process that takes place during this stage is based on static/dynamic rules (like firewall rules).

**Rate Limiting**: Based on known traffic patterns, signatures can be used to detect the rate of incoming calls. For example, velocity and acceleration values (first and second order derivative) of the number of arriving calls from a given user/host/domain can be used as a detection mechanism. That is, when the velocity/acceleration reaches a certain threshold, the drop rate can be updated through feedback control. As expected, the sooner we detect a change in the incoming pattern based on signatures, the faster there will be a reduction in the spread of the spam. Once spamming is identified, PID (Proportional Integral Control) feedback control can be used to reduce the velocity of spreading. This method of detection is useful not only in deterring spamming attacks but also in DOS (denial of service, where large number of messages sent in a short period of time) attacks. All the results have been discussed in[5].

**Black and White Lists:** Most of the spam detection is done using a set of valid and invalid signatures. These signatures would make the Spam detection server know which calls the server has to forward and which calls the server has to block. This is a direct way of quarantining the calls where the end user would specify a set of entities from which it is always ready to receive calls encoded in white lists and a different set of entities from which it would like to see all calls being blocked that are encoded in Blacklists. The entities might be any of the end user or an end soft-phone or a domain. Depending upon the specification of the end user, the specified calling users would be allowed or denied calling. The lists are customized. i.e. each end user would have the flexibility of specifying its own entries. Entries differ in each of the lists specified by different end users and thus would have no bearing whatsoever of influencing the call forwarding or blocking of other end users. i.e. each end user would be guaranteed of forwarded or denied calls based on its own customized list. The Voice Spam Detector would let forward all the calls from the trusted elements in the white lists and block all the calls from un-trusted elements in the blacklist.

The black and white lists are constructed using user feedback to the VSD. When after forwarding the call, the user responds with a spam feedback message saying that the present call was a spam call to the VSD, the VSD adds the new entry to the black list and any future call with the same parameters is directly blocked at the server and is not forwarded. On the other hand, if the user specifies that the present call is important to it and want to receive any such calls in the future with these parameters, the entry is added to the user white list and any future calls with the same parameters are directly forwarded to the end user.

**Bayesian Learning:** Learning the behavior of the participating entities would let us make many intelligent decisions regarding the call. The behavior of the participating entities can be learnt during the course of a period of time. The behavior can be estimated by their past history of calling to the called party's domain end users. This process of observing the calling party's behavior over a period of time is termed as Learning. Learning as such represents an abstract modeling of the calling party's past behavior. The observed behavior over the period of time would classify the participating entities as spam producing or valid.

For an incoming call, the VSD would examine the participating entities like the call source (end user, host, domain etc.), participating proxies in routing etc with the help of fields like "from", "to", "record

route", "via". VSD checks for any spam behavior associated with any of the participating entities by looking up trust information available for those entities. The trust information would be available if any of the entities has a history of calling an end user in the analysis domain. The spam probability of the call(i.e., associated trust level of the call) can be computed using Bayesian inference techniques [1]. The spam probability of an incoming call is P(X | C = spam) for a message X= $\{x_1,x_2,x_3 \ldots x_n\}$ and can be calculated by

$$\frac{P(C = spam) \prod_{i=1..n} P(x_i = spam)}{\sum_{k=spam,valid} P(C = k) \prod_{i=1..n} P(x_i = k)}$$

where each of $x_1..x_n$ represent different identifiers in the header of a signaling message, like "From", "To", "Via" "Record Route", and "Contact Info"). VSD would be filtering out the calls if the spam probability of the call would be greater than the permissible limit or tolerance level. Otherwise, the call is forwarded to the actual recipient of the call and the VSD waits for a feedback from the recipient. All the call processing depends on the end users reaction on the just forwarded call. The recipient responds with a message about the nature of the call. If the recipient responds with a message saying that the present call is a spam call, the VSD logs the call source information for future spam analysis. Future calls with any of the above participating entities would have a high degree of spam probability and more chances of getting stopped at VSD. On the other hand, if the recipient responds with a valid call message, the trust of the participating entities is updated to depict more valid probability i.e. less spam probability. Often, the called party does not know the calling party and hence there is no history of trust for a specific caller. In this context, we can infer the reputation of the calling party by using social networks.

The permissible limit or tolerance level is chosen by giving a preference of valid calls over spam calls i.e. the number of spam messages that can be let in so as to minimize the blocking of valid calls called "False Positives". The main aim of any spam filtering technique should be to minimize the "False Negatives"(spam calls let in as valid) keeping the false positives to zero. This ratio of valid calls to permitted spam calls would give a measure of the permissible limit. And any call that exceeds the permissible limit would be classified as spam and quarantined.

**Social Networks and Reputation:** Social networks can be used to represent user relationships that can be derived along the paths of the network. These relationships are transitive and transparent [4]. If Alice is related to Bob and Bob is related to Charles, then with a high degree of confidence, an argument can be made that Alice is related to Charles. These social networks can be used to infer the associated relations between the social elements. With respect to a VoIP service user, the user's social network represents the associated and trusted neighbors from which the user is willing to receive calls.

With respect to a proxy, a graph can be generated using the neighboring proxies and their users. Subsequently this graph can be used in deriving the reputation of a calling party. Reputation implies social understanding. Reputation is derived from trusted peers (e.g., nearest proxies reachable in one hop) while the trust is calculated based on the past history. The peer proxies would derive reputation by their trusted peer proxies, and this would continue until the last proxy in the "via" list or the proxy that is reachable from source by one hop is reached. Based on the reputation inference from the peer proxies of the source and the entities in-between, the reputation can be inferred. If R (a,b) gives a's reputation on c, $R(a,c) = \Theta(R(a,b),R(b,c))$ for all (b) in trust-neighbors of (a). We believe that $\Theta$ is a Bayesian inference function on the proxies bearing the topology depicted by the graph. For a given call to an end user in the receiving domain, the reputation of the domain from which the call originated is inferred and the spam probability of the call obtained by trust inference is updated based on reputation inference. If the call is let through VSD to the receiving end user, based on the feedback given by the end user to the VSD, the reputation of the call originating domain and all intermediate domains that have routed the call are updated. The update is positive for a valid call and negative for a spam call.

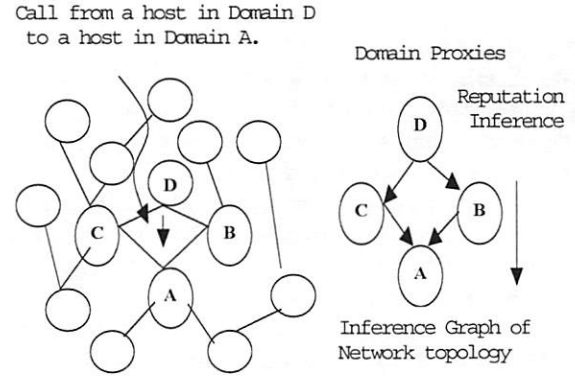



**Fig 2:** Reputation Inference for a call from Domains D to A

*Many a times, trust and reputation are used for representing human belief. Trust represents caller's past behavior while reputation signifies social*

*status. While trust is a calculated entity, reputation is derived. Reputation is inferred by modeling human behavior.* i.e., in [Fig 2], if there was a spam call from domain D to domain A through domain C, then the reputation of D is decreased and also the reputation of C is decreased for forwarding a spam call. For a second spam call from domain D to domain A, the decrease in reputation is more than compared to the decrease for the first spam call. Also, the decrease in reputation for a spam call is more than the increase in reputation for a valid call. We achieve this by increasing the reputation additively for a valid call and decreasing multiplicatively for a spam call i.e. *an additive increase and multiplicative decrease in reputation*. In this way, using reputation and trust from past history, the calls can be quarantined or classified as spam.

For the given topology graph in [Fig 2], reputation is inferred by using Bayesian networks. For a call form domain D to domain A, the reputation can be inferred by calculating P(A|D) i.e. the posterior probability of A given an event that a call has been generated at D.

$$P(A|D) = P(A,B|D) + P(A,\sim B|D)$$
$$= P(A|B)P(B|D) + P(A|\sim B)P(\sim B|D) \quad \text{Eq 1}$$

where $P(A|B) = P(A,C|B) + P(A, \sim C|B)$

$$= P(A|B,C)P(C) + P(A|B\sim C)P(\sim C) \quad \text{Eq 2}$$

and $P(A|\sim B) = P(A,C|\sim B) + P(A, \sim C|\sim B)$ Eq 3

$$= P(A|\sim B,C)P(C) + P(A|\sim B, \sim C)P(\sim C)$$

$$P(C) = P(C,D) + P(C, \sim D) \quad \text{Eq 4}$$
$$= P(C|D)P(D) + P(C|\sim D)P(\sim D)$$

Solving equations 1-4 gives the updated probability or updated reputation of D. For a given set of initial or prior probabilities to the nodes of the topology graph representing the reputation of those domains, for a spam call from domain D to domain A the Bayesian inference calculations shown above would decrease the reputation for B,C and D proxies, and increase the reputation for a valid call for the same.

## 4. Experimental Setup and Results

The experimental setup consists of the Voice spam detection server, the end users for whom the VSD is acting as a spam detector and the call generating domains from which calls would be generated to the end users in the receiving domain. The end clients in the calling domain and the called domain are simulated SIP soft clients strictly in compliant with SIP RFC[11]. *All the simulated clients and the Voice Spam Detector are compatible with the real SIP phones and are capable of establishing sessions with them.* The simulated clients on the call-generating end generate calls by using randomly chosen usernames and hosts in the SIP URI of "from" field. The call generation process uses a Bernoulli distribution and the calls are generated with an average rate of 8 calls/minute. Neither the VSD nor the called domain end users have any idea regarding the call generation process. *A button called "SPAM" included in each IP phone in the receiving domain to give feedback to the VSD.*

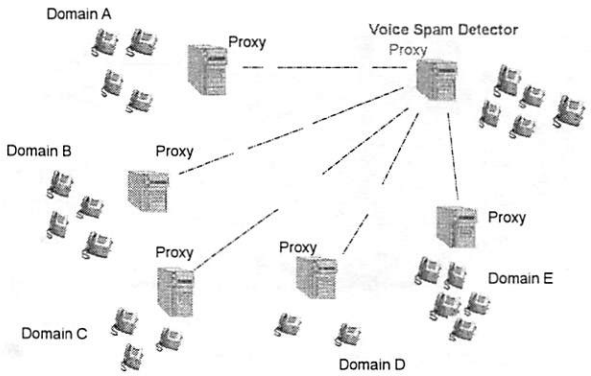



**Fig 4:** Experimental Setup showing the Calling and Receiving domains.

The simulated end clients on the call generating end randomly generate calls to the VSD and the VSD analyzes the call based on the caller trust and reputation. VSD calculates the spam probability of the call and compares with a predetermined threshold value to infer the spam behavior and block them. The threshold values chosen for each stage of analysis depends upon factors like the learning period, minimization of false alarms (false positives and false negatives) etc. Learning period signifies the minimum number of calls required by the VSD to learn the spam behavior before it starts blocking spam calls.

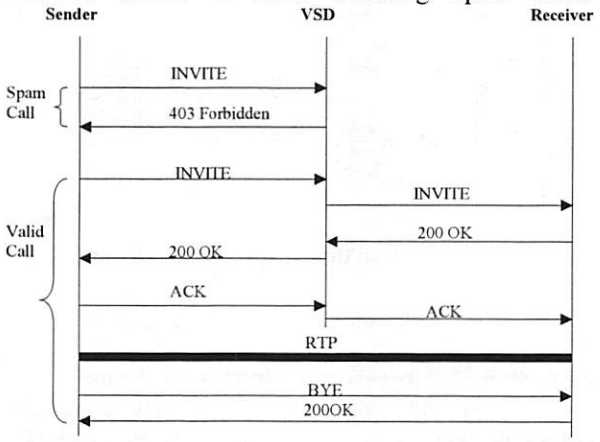



**Fig 3:** Call Flow for Spam and Valid calls through the VSD

The called domain users are equipped with spam recognition capabilities. We configure the calling domain with randomly chosen set of users, hosts and

domains as spammers before the start of the experiment. The call received by the receiving client is analyzed and a feedback is given to the VSD about the nature of call. The Voice Spam Detector learns by observing the calling pattern with respect to called users, hosts and domains and the received feedback.

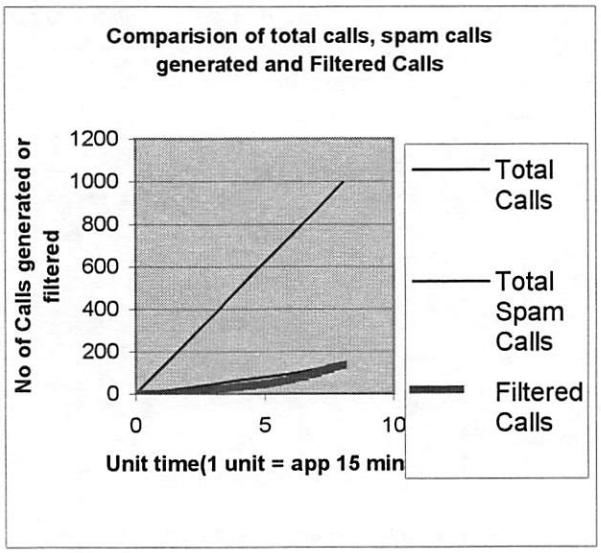



**Fig 5:** Comparing the actual calls generated, actual spam calls generated and filtered calls.

[Fig 5] represents the comparison between total calls, total spam calls generated and number of spam calls blocked by the VSD. The results are shown for five calling domains with each domain having an average of 100 users and 35 hosts. The number of calls blocked is a result of all the three stages of analysis [See Sec 3.2]. i.e. the black and white listing, trust (past history) and reputation of the calling party.

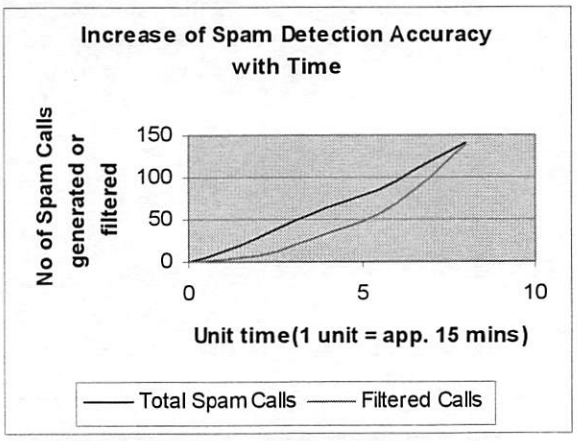



**Fig 6:** Spam Detection Accuracy increases with time.

[Fig 6] represents the comparison between the spam calls generated and filtered calls by the VSD. Initially VSD has no knowledge of spam generating clients, but learns the spam behavior with time and feedback from the end users. The spam calls detected are equal to the actual spam calls generated after certain learning period with an accuracy of 97.6% and a false positive percentage of 0.4%.

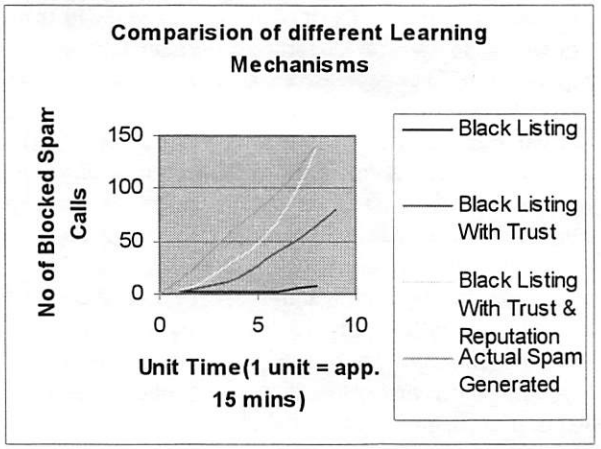



**Fig 7:** Spam Calls blocked by VSD for different stages of analysis.

[Fig 7] shows the spam calls blocked for the three stages of analysis. The experiments are conducted with a random 100 users and 35 hosts in each of 5 domains on the call generating end. It can be observed that the number of spam calls blocked using blacklisting, trust and reputation is approximately 97.16% compared to 4.25% if only blacklisting is implemented.

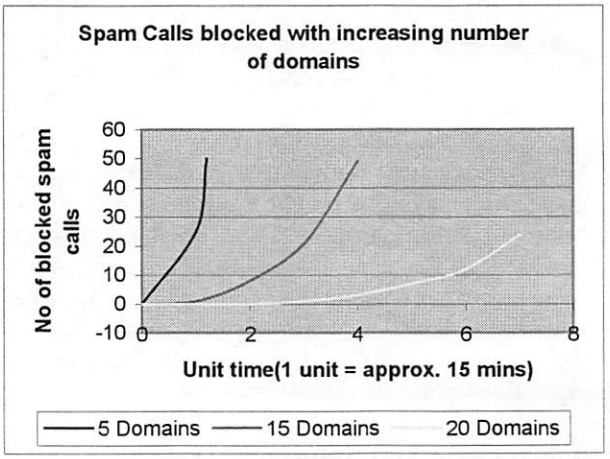



**Fig 8:** Blocked Spam Calls for increasing scalability on call generation.

[Fig 8] gives the number of spam calls blocked for three different sizes of topology. The time taken by VSD for learning spam behavior from 20 domains is more compared to time taken for 15 domains for the same set of spammers i.e. for the same set of spam users, hosts and domains. However, the VSD would have the near about the same approximate accuracy of spam recognition when the number of spammers increase with increase in the number of call generating users, hosts and domains. For the analysis shown in [Fig 8], the false alarms and the accuracy of VSD is as shown in [Tab 1].

| # of Domains | Filter Accuracy % | False Positives % | False Negatives % |
|---|---|---|---|
| 5 | 97.6 | 0.4 | 2 |
| 15 | 97.52 | 0.19 | 2.3 |
| 20 | 97.595 | 0.11 | 2.36 |

**Tab 1**: False Alarms and Spam recognition accuracy for the analyzed calls in Fig 8.

## 5. Conclusion

It is estimated that 35 billion spam email messages per day were generated in 2004. These messages are nuisance to the receivers and in addition create low availability and network congestion. VoIP technology is replacing existing PSTN at a rapid pace. The problem of spam in VoIP networks has to be solved in real time compared to e-mail systems. Many of the techniques devised for e-mail spam detection rely upon content analysis and in the case of VoIP it is too late to analyze the media after picking up the receiver. So we need to stop the spam calls before the telephone rings.

In computing, trust has traditionally been a term relating to authentication, security, or a measure of reliability. When it comes to receiving or rejecting a voice call social meaning of trust is applied and in particular reputation of the calling party is analyzed. We developed a five-stage process for identifying if the incoming call is spam or not. These stages include multivariable Bayesian analysis and inferring reputation using Bayesian networks. The results from each stage are fed back for collaboration between different processes. We have verified the results using an experimental setup consists of more than randomly generated calls from several thousand soft clients and a SIP proxy server. This setup includes commercial grade proxy server software as well as soft client. We have added the spam filter software at the proxy server for preventing and detecting the spam. We found that combining black/white lists, trust of the calling party and reputation of the calling party can be used accurately to identify if it is a spam or not. In this analysis we have used a concept where trust can be built up over time but a single spam call can exponentially bring down the trust level. We used this concept and found that the call can be more accurately identified as spam after a period of learning. From our observation of the logs it takes at least 3 spam calls to confirm it is a spam and fourth call can be accurately identified as the spam. Finally we expanded the experiments with large number of domains and verified our filtering mechanism. Further work involves understanding the behavior of different controlling parameters in trust and reputation calculations and deriving meaningful relationships between them. Also, we believe that our multistage filtering architecture can be used in prevented unwanted emails as well as in electronic commerce.

## References


1. M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. 1998. A Bayesian Approach to Filtering Junk E-Mail. *Learning for Text Categorization – Papers from the AAAI Workshop*, pages 55–62, Madison Wisconsin. AAAI Technical Report WS-98-05.
2. G. Salton, M.J. McGill. 1983. Introduction to Modern Information Retrieval. McGraw- Hill.
3. T.M. Mitchell, Machine Learning. McGraw-Hill, 1997.
4. J. Golbeck, J. Hendler, "Reputation Network Analysis for Email Filtering", IEEE conference on Email and Anti Spam, August 2004.
5. R. Dantu, J. Cangussu, A. Yelimeli, "Dynamic Control of Worm Propagation", IEEE International Conference on Information Technology ITCC April 04
6. N. Soonthornphisaj, K. Chaikulseriwat, P Tang-On, "Anti-Spam Filtering: A Centroid Based Classification Approach", IEEE proceedings ICSP 02
7. G. Sakkis, I. Androutsopoulos, G. Paliouras, V. Karkaletsis, C.D. Spyropoulos, P. Stamatopoulos, "A memory based approach to anti-spam filtering for mailing lists", Information Retrieval 2003.
8. W.W. Cohen, "Learning Rules that Classify e-mail", In Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access, 1996.
9. P.O. Boykin, V. Roychowdhury, "Personal email networks: an effective anti-spam tool". Preprint, http://www.arxiv.org/abs/cond-mat/0402143, (2004).
10. I Rigoutsos, T. Huynh, "Chung-Kwei: A Pattern Discovery based System for the Automatic Identification of Unsolicited E-mail messages", Proceedings of the first conference on E-mail and Anti-Spam, 2004.
11. J. Rosenberg, H Shulzrinne, G Camerillo, A Johnston, J Peterson, R Sparks, M. Handley, E. Schooler, "Session Initiation Protocol", RFC 3261, June 2002.
12. http://www.forbes.com/technology/2004/02/27/cx_ah_0227tentech.html

# The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets

Evan Cooke,* Farnam Jahanian,*† Danny McPherson†
*Electrical Engineering and Computer Science Department †Arbor Networks
University of Michigan *danny@arbor.net*
*{emcooke, farnam}@umich.edu*

## Abstract

Global Internet threats are undergoing a profound transformation from attacks designed solely to disable infrastructure to those that also target people and organizations. Behind these new attacks is a large pool of compromised hosts sitting in homes, schools, businesses, and governments around the world. These systems are infected with a *bot* that communicates with a bot *controller* and other bots to form what is commonly referred to as a *zombie army* or *botnet*. Botnets are a very real and quickly evolving problem that is still not well understood or studied. In this paper we outline the origins and structure of bots and botnets and use data from the operator community, the Internet Motion Sensor project, and a honeypot experiment to illustrate the botnet problem today. We then study the effectiveness of detecting botnets by directly monitoring IRC communication or other command and control activity and show a more comprehensive approach is required. We conclude by describing a system to detect botnets that utilize advanced command and control systems by correlating secondary detection data from multiple sources.

## 1 Introduction

Global Internet threats are undergoing a profound transformation from attacks designed solely to disable infrastructure to those that also target people and organizations. This frightening new class of attacks directly impacts the day-to-day lives of millions of people and endangers businesses around the world. For example, new attacks steal personal information that can be used to damage reputations or lead to significant financial losses. Current mitigation techniques focus on the symptoms of the problem, filtering the spam, hardening web browsers, or building applications that warn against phishing tricks. While tools such as these are important, it is also critical to disrupt and dismantle the infrastructure used to perpetrate the attacks.

At the center of these threats is a large pool of compromised hosts sitting in homes, schools, businesses, and governments around the world. These systems are infected with a *bot* that communicates with a bot *controller* and other bots to form what is commonly referred to as a *zombie army* or *botnet*. A bot can be differentiated from other threats by a communication channel to a controller. Many bots found in the the wild today are a hybrid of previous threats combined with a communication system. They can propagate like worms, hide from detection like many viruses, and include attack methods from toolkits.

The magnitude of the botnet problem is just beginning to be carefully documented. According to a recent report, the number of new bots observed each day rose from less than 2,000 to more than 30,000 over the first six months of 2004 [7]. The total number of bot infected systems has been measured to be between 800,000 to 900,000 and CERT has described botnets with more than 100,000 members [12, 6].

Botnets are a very real and quickly evolving problem that is still not well understood. In this paper, we outline the problem and investigate methods of stopping bots. We identify three approaches for handling botnets: (1) prevent systems from being infected, (2) directly detect command and control communication among bots and between bots and controllers, and, (3) detect the secondary features of a bot infection such as propagation or attacks.

The first approach is to prevent systems from being infected. There are a range of existing techniques, including anti-virus software, firewalls, and automatic patching.

The second approach is to directly detect botnet command and control traffic. Botnets today are often controlled using Internet Relay Chat (IRC) and one possible method of detecting IRC-based botnets is to monitor TCP port 6667 which is the standard port used for IRC traffic [9]. One could also look for non-human behavioral characteristics in traffic, or even build IRC server scanners to identify potential botnets [17, 19].

We argue there is also a third approach that detects botnets by identifying secondary features of a bot infection such as propagation or attack behavior. Rather than directly attempting to find command and control traffic, the key to this approach is the correlation of data from

different sources to locate bots and discover command and control connections.

In this paper we investigate the second and third approach for stopping botnets. The problem with the first approach is that preventing all systems on the Internet from being infected is nearly an impossible challenge. As a result, there will be large pools of vulnerable systems connected to the Internet for many years to come.

The paper begins by describing how the botnet problem is evolving by tracing the origins of bots. We then demonstrate the size of the botnet problem using evidence from the operator community, the Internet Motion Sensor project [2], and experimental data collected from a honeypot experiment. With this information, we examine current IRC-based botnet communication and detection strategies. Next, we show how finding botnets by detecting command and control messages will become less effective as attackers move to other communication topologies and obfuscate their communications. We conclude by describing a system to identify botnets by correlating secondary detection data with host-based forensic information.

## 2 Bots

Studying the evolution of bots and botnets provides insight into their current capabilities. One of the original uses of computer bots was to assist in Internet Relay Chat (IRC) channel management [16]. IRC is a chat system that provides one-to-one and one-to-many instant messaging over the Internet. Users can join a named channel on an IRC network and communicate with groups of other users. Administering busy chat channels can be time consuming, and so channel operators created bots to help manage the operation of popular channels. One of the first bots was Eggdrop, which was written in 1993 to assist channel operators [1].

In time, IRC bots with more nefarious purposes emerged. The goal of these bots was to attack other IRC users and IRC servers. These attacks often involved flooding the target with packets (i.e., DoS attacks). The use of bots helped to hide the attacker because the attack packets were sent from the bot rather than directly from the attacker (assuming a non-spoofed attack). This new level of indirection also allowed multiple computers to be grouped together to perform distributed attacks (DDoS) and bring down bigger targets.

Larger targets required more bots, and so attackers looked for methods to recruit new members. Since very few users would agree to have their computers utilized for conducting packet floods, attackers used trojaned files and other surreptitious methods to infect other computers. For example, bots such as SubSeven Bot, Bionet Bot, Attack Bot, GTBot, EvilBot, and Slackbot are often simple to install remotely or hide in potentially legitimate files [10].

As the economic incentives to use bots for DoS extortion, spam, phishing and other attacks have emerged, the bot infection process has become more automated. For example, SDBot [11] (also known as rBot) can propagate using many different mechanisms such as open file shares, p2p networks, backdoors left by previous worms, and exploits of common Windows vulnerabilities such as WEBDAV [14], DCOM RPC [13], and LSASS [15]. The attack and communication capabilities of modern bots have also become extremely advanced. For example, Agobot [4] (also know as Phatbot or Gaobot) has a large range of built-in attack capabilities including denial of service attacks, a proxy for spam, GRE tunneling, and password sniffers.

In many respects, the bots found in the the wild today are a hybrid of many previous threats integrated with a command and control system. They can propagate like worms, hide from detection like many viruses, and include attack methods from toolkits. Of even greater concern, the construction of bots is now very much a cooperative effort. An example is the source code of SDBot which contains comments from many different authors. The result is a proliferation of different bot variants. As of August 2004, SDBot has been reported to have more than 4,000 variants [11].

## 3 Botnet Measurements

As bots have evolved, evidence has emerged suggesting the number of bot infection has grown dramatically. In this section we show the growing problem from the perspective of the operator community, using data from the Internet Motion Sensor project, and using data generated by a honeypot experiment.

### 3.1 Operator Experiences

As criminals update their tools for the digital age, they have turned to bots as a weapon of choice. Those that run digital networks are caught between the attackers and their targets, but also have a unique perspective on the situation.

To better understand bot behavior, we informally interviewed five major backbone operators at Tier-1 and Tier-2 providers. They indicated that the botnet problem is very real, and something they combat frequently. They also provided interesting insight into current botnet trends.

While the number of botnets appears to be increasing, the number of bots in each botnet is actually dropping. A few years ago, botnets with 80k to 140k members were observed [6]. Today, botnets with a few hundred to a few thousands hosts are common. There are several factors that may be driving this trend. First, smaller botnets are more difficult to detect and may be easier to sell or rent.

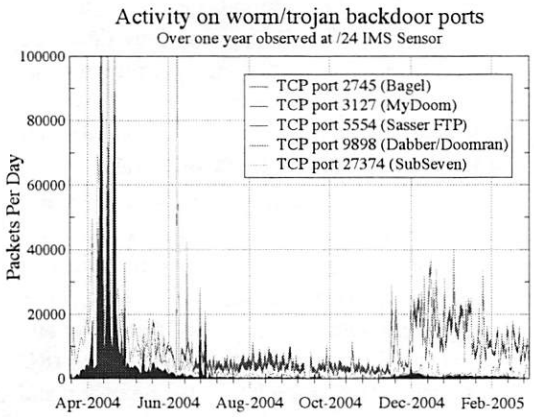



Figure 1: Backdoor activity over one year as observed by a /24 IMS sensor

Another major consideration is the additional *firepower* of to each bot due to the proliferation of DSL, cable, and other broadband access technologies. For example, only a few hundred hosts having a cable broadband Internet connection with an upstream bandwidth of 1Mbps can saturate a high-speed OC3 (155 Mbps) Internet link used by large businesses.

What is clear is that botnets have become a business opportunity, and the characteristics of botnets today often correspond to economic considerations. For example, botnet controllers have been observed building botnets-to-order. These custom botnets might consist of systems within educational networks or, more frighteningly, of systems from government networks.

## 3.2 Botnet Propagation

Accurately measuring the number of active botnets is very difficult because there are few overt characteristics to identify. One method of tracking bots is to look for propagation activity. The mechanisms used by bots to spread vary greatly, but one major mechanism is to scan for vulnerabilities. These weaknesses can be in production software or in other malware. For example, the Bagel and MyDoom worms left backdoors that could be used to run arbitrary code. These backdoors are known infection vectors for bot activity. Starting in April 2004, the Internet Motion Sensor project (IMS) observed a large uptick in activity on these backdoors. The IMS is a network of sensors that monitors 60 unused address blocks at 19 diverse organizations [2]. Figure 1 shows the number of packets received at well-known backdoor ports left by worms and trojans over a one year period at a /24 (256 address wide) IMS sensor. Figure 1 demonstrates the large amount of continued activity on these port.

## 3.3 Bot Honeypot

To get a better understanding of bots firsthand, we set up an experiment to measure botnets on a real network. The idea was very simple. If botnets are such a serious problem, then a vulnerable system should be quickly recruited into a botnet when placed on the Internet [5, 19]

The experimental setup was as follows. We placed a new system with a fresh installation of Windows 2000 and XP without any service packs (i.e. a honeypot) behind a transparent proxy device (FreeBSD bridge). The proxy performed three operations: (1) it rate limited traffic in and out to 12KB/s; (2) it disallowed access to systems on the local network; and (3) it logged all traffic to and from the vulnerable system. The proxy was carefully monitored to ensure that the honeypot did not perform any illicit activities such as sending spam or participating in attacks.

We performed 12 experimental runs in which each run was typically 12-72 hours long and traces typically topped 100MB of activity. The traces contained repeated compromises achieved using a wide range of old vulnerabilities, including the DCOM RPC [13] vulnerability and the LSASS [15] vulnerability. The honeypot was often recruited in multiple botnets at the same time.

This experiment suggests a widespread bot problem. Over the 12 runs, only 2 included an infection from a worm (i.e. malware without command and control). While this single deployment may not be representative of other networks, when combined with the experiences of the operational community and the IMS data, the evidence suggests a significant evolution in the threat landscape has already occurred.

# 4 Botnets Today: Detecting Command and Control

To combat the growing problem of bots, we identified two approaches for detecting botnets: detect the command and control communication, or detect the secondary features of a bot infection. In this section we study methods of detecting botnets by directly locating command and control traffic.

## 4.1 IRC-based Command and Control

A bot must communicate with a controller to receive commands or send back information. One method for establishing a communication channel is to connect directly to the controller. The problem is that this connection could compromise the controller's location. Instead, the bot controller can use a proxy such as a public message drop point (e.g., a well-known message board). However, because websites and other drop points can introduce significant communication latency, a more active approach is desirable. A well-known public ex-

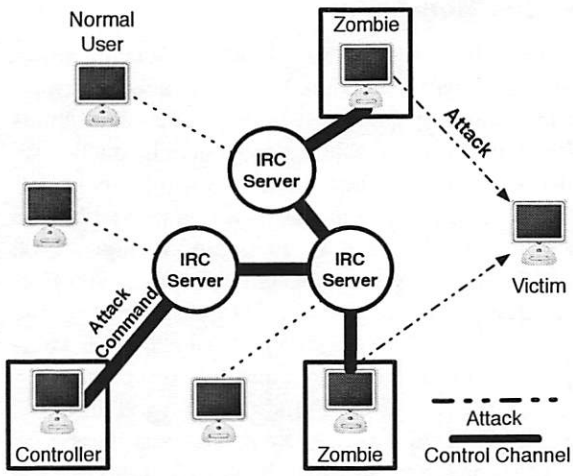



Figure 2: IRC-based botnet DDoS Attack

change point that enables virtually instant communication is IRC.

IRC provides a common protocol that is widely deployed across the Internet and has a simple text-based command syntax. There are also a large number of existing IRC networks that can be used as public exchange points. In addition, most IRC networks lack any strong authentication, and a number of tools to provide anonymity on IRC networks are available. Thus, IRC provides a simple, low-latency, widely available, and anonymous command and control channel for botnet communication.

An IRC network is composed of one or more IRC servers as depicted in Figure 2. In a typical botnet, each bot connects to a public IRC network or a hidden IRC server on another compromised system. The bot then enters a named channel and can receive commands directly from a controller or even from sequences encoded into the title of the channel. The bot and any other bots in the same channel can then be instructed to attack as shown in Figure 2.

## 4.2 IRC-based Botnet Detection

Today, most known bots use IRC as a communication protocol, and there are several characteristics of IRC that can be leveraged to detect bots. In this section, we describe methods of detecting IRC-based botnets.

One of the simplest methods of detecting IRC-based botnets is to offramp traffic from a live network on known IRC ports (e.g., TCP port 6667) and then inspect the payloads for strings that match known botnet commands. Unfortunately, botnets can run on non-standard ports. We detected at least three such botnets running on high-numbered ports in our honeypot experiment.

Another method is to look for behavioral characteristics of bots. One study found that bots on IRC were idle most of the time and would respond faster than a human upon receiving a command. The system they designed looked for these characteristics in Netflow traffic and attempted to tag certain connections as potential bots [17]. The approach was successful in detecting idle IRC activity but suffered from a high false positive rate.

Given problems such as false positives on live networks, another approach is to use a non-productive resource or *honeypot*. One group set up a vulnerable system and waited for it to be infected with a bot. They then located outgoing connections to IRC networks and used their own bot to connect back and profile the IRC server [19]. However, they did not take the next step and develop a detection system based on the technique.

Rather than connecting to the IRC server directly, another approach is to use a honeypot to catch the bot and then look for characteristics of command and control traffic in the outgoing connections. Using the data collected in our honeypot experiment described in Section 3, we attempted to isolate behavioral invariants in botnet communication. We located all successful outgoing TCP connections and verified that they were all directly related to command and control activity by inspecting the payloads. There were a wide range of interesting behaviors, including connections from the bot to search engines to locate and use bandwidth testers, downloading posts from popular message boards to get server addresses, and the transmission of comprehensive host profiles to other servers. These profiles included detailed information on the operating system, host bandwidth, users, passwords, file shares, filenames and permissions for all files, and a number of other minute details about the infected host.

We then analyzed all successful outgoing connections and looked for specific characteristics that could be used to identify botnet command and control traffic. The results suggested that there are no simple characteristics of the communication channels themselves that can be used for detection. For example, the length of the outgoing connections varied widely, with certain connections lasting more then 9 hours and others less than a second. The number of bytes transfered per connection also varied widely even when we separated out IRC communication from other command and control activity.

The results from our analysis nor the results from previous bot detection efforts has revealed any simple connection-based invariants useful for network detection. One might inspect every payload of every packet however this is currently very costly on high throughput networks. More importantly, attackers can make small modifications that make detection nearly impossible. For example, encrypting traffic, masking flow behavior with random noise, and even switching to different communication topologies can make detection im-

| Topology | Design Complexity | Detectability | Message Latency | Survivability |
|---|---|---|---|---|
| Centralized | *Low* | *Medium* | *Low* | *Low* |
| Peer-to-Peer | *Medium* | *Low* | *Medium* | *Medium* |
| Random | *Low* | *High* | *High* | *High* |

Table 1: Command and Control Topologies

mensely more challenging (versions of AgoBot with SSL encryption have been reported). In the end, any approach that relies on directly detecting command and control traffic can be defeated by changing the mode or behavior of the communication.

## 5 Botnets of Tomorrow

The difficulty in capturing bot command and control illustrates the need for a more robust detection approach. In this section we investigate possible advanced bot communication topologies and then present a detection method that gets around these difficulties by identifying other characteristics of a bot infection.

### 5.1 Command and Control Models

To explore the implications of various bot communication methods, we identify three possible topologies and investigate their associated benefits and weaknesses as shown in Table 1.

**Centralized:** A centralized topology is characterized by a central point that forwards messages between clients. Messages sent in a centralized system tend to have low latency as they only need to transit a few well-known hops. From the perspective of an attacker, centralized systems have two major weaknesses: they can to be easier to detect since many clients connect the same point, and the discovery of the central location can compromise the whole system.

**P2P:** Peer-to-peer (p2p) botnet communication has several important advantages over centralized networks. First, a p2p communication system is much harder to disrupt. This means that the compromise of a single bot does not necessarily mean the loss of the entire botnet. However, the design of p2p systems are more complex and there are typically no guarantees on message delivery or latency. A structured p2p location service such as Chord [18] could be used, but such a system might also reveal compromising information about other nodes. Existing p2p anonymity networks could be adapted, however there would be additional processing and latency overhead [3].

**Random:** A botnet communication system could also be based on the principle that no single bot knows about any more than one other bot. In this topology a bot or controller that wanted to send a message would encrypt it and then randomly scan the Internet and pass along the message when it detected another bot. The design of such a system would be relatively simple and the detection of a single bot would never compromise the full botnet. However, the message latency would be extremely high, with no guarantee of delivery. In addition, the random probing behavior could be detectable.

The three topologies described above can be viewed as a spectrum of information release. That is, they describe the maximum number of nodes any one node will know about at one time. In a central topology the server knows about all nodes, while in a random topology no node ever knows more than one other node. Each topology has specific advantages and drawbacks, and the optimal topology for a given botnet might exist somewhere between the extremes. Many existing communication systems may fall somewhere in between. IRC could be classified as a centralized system, although the server-to-server and client-to-client communication is more like peer-to-peer systems.

The implication of this analysis is that command and control communication is extremely flexible, and a bot could use any number of different channels and different topologies to communicate. Thus, it is difficult for any general botnet detection scheme to rely on specific communication characteristics.

### 5.2 Advanced Botnet Detection

In the end, all methods that rely on particular communication protocols or topologies like IRC will lose effectiveness as attackers modify their tools. For these reasons, we argue that long term efforts to stop botnets should focus on other methods. One approach is to combine data from existing proven detection systems to identify suspect activity.

Such a system could use data from host detectors, network detectors, or a combination of both. The detector could monitor a production resource or a non-productive resource (i.e. *honeypot*). A key requirement of this approach is the ability to aggregate and summarize data from heterogeneous sources. For example, a system could use a network detector to provide an alert on noisy behaviors such as scanning or DoS activity. The alert could then be traced back to the host that initiated the activity. Using a host-based monitor, the packets could be correlated with the sending process, and the bot program identified [8]. Finally, using the the same host monitor, other other suspected command and control channels related to that process could be identified.

There are clearly a number of challenges in realizing this approach. As a first step, we intend to identify the different available sources of detection data and then evaluate effective trace-back mechanisms on the host. Although more complicated than just identifying command and control messages, we believe a multi-detector correlational approach will provide a more robust and longer-term botnet detection system.

### 5.3 Challenges of Botnet Disruption

Detection is not the only step involved in stopping a botnet. Given the detection data, an action plan to disrupt the botnet must be formulated. We now describe a summary of the challenges involved in botnet disruption. When a bot is detected, there are two immediate mitigation goals: taking down the bot and taking down the botnet. A bot is different from a worm or virus because it is a member of a larger botnet and there is significant value in taking down the whole botnet rather than just a single node. The process is analogous to law enforcement efforts to capture a gang rather than a single person.

The capture of a single gang member can provide information about a whole gang, and so it may be useful to let that person operate for a time to inform on other members. For bots today, that process is straightforward because many bots communicate with a single IRC server. However, as bots evolve it is likely that it will become increasingly difficult to identify other members of a botnet.

Continuing the law enforcement analogy, gangs that cross international boundaries require the coordination of law enforcement in different countries. Similarly, botnets can be highly distributed, with nodes in hundreds of networks located in many different countries. Stopping botnets will require a significant level of cooperation among providers and some level of automation.

## 6 Conclusion

This paper has outlined the origins and structure of bots and botnets, and shown how they have evolved to become potent weapons. We studied methods of detecting IRC-based bots and demonstrated three general command and control topologies to illustrate the difficulty of focusing detection efforts on command and control traffic. Based on this understanding, we described an approach to detect botnets by correlating secondary detection information to pinpoint bots and botnet communication.

The threat landscape on the Internet is undergoing an important transformation, and researchers and practitioners need to adapt to the new covert, distributed, and global nature of the threat. This requires collaboration among researchers to devise hybrid data analysis techniques and collaboration between network operators to more quickly and automatically act on threats. Botnets are a global problem that affects the entire Internet community and requires a community effort to stop them.

## Acknowledgments

This work was supported in part by the Advanced Research and Development Activity (ARDA). The Internet Motion Sensor project is supported by the Department of Homeland Security, Intel, and Cisco. Many thanks to David Watson, Michael Bailey, Jose Nazario, Tim Battles, Nicolas Fischbach, Chris Morrow, and Rob Thomas for helpful comments and feedback. We would also like to thank all the IMS participants.

## References

[1] Eggdrop: Open source IRC bot. `http://www.eggheads.org/`, 1993.

[2] Michael Bailey, Evan Cooke, Farnam Jahanian, Jose Nazario, and David Watson. The Internet Motion Sensor: A distributed blackhole monitoring system. In *Proceedings of Network and Distributed System Security Symposium (NDSS '05)*, San Diego, CA, February 2005.

[3] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[4] Computer Associates. Win32.Agobot. `http://www3.ca.com/securityadvisor/virusinfo/virus.aspx?id=37776`, July 2004.

[5] Nicolas Fischbach. Router forensics DDoS/worms update. `http://www.securite.org/`, 2003.

[6] Allen Householder and Roman Danyliw. CERT Advisory CA-2003-08 Increased Activity Targeting Windows Shares. 2003.

[7] Kim Legelis, Symantec Corporation. Combating online fraud: An update. `http://information-integrity.com/article.cfm?articleid=100`, 2005.

[8] Samuel T. King, Z. Morley Mao, Dominic G. Lucchetti, and Peter M. Chen. Enriching intrusion alerts through multi-host causality. In *Proceedings of Network and Distributed System Security Symposium (NDSS '05)*, San Diego, CA, February 2005.

[9] John Kristoff. Botnets. 32nd Meeting of the North American Network Operators Group, October 2004.

[10] LockDown Corp. Bots, Drones, Zombies, Worms and other things that go bump in the night. `http://swatit.org/bots/`, 2003.

[11] McAfee. W32/Sdbot.worm. `http://vil.nai.com/vil/content/v_100454.htm`, April 2003.

[12] Laurianne McLaughlin. Bot software spreads, causes new worries. *IEEE Distributed Systems Online*, 5(6), June 2004.

[13] Microsoft. DCOM RPC vulnerability. `http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx`, July 2003.

[14] Microsoft. WEBDAV vulnerability. `http://www.microsoft.com/technet/security/bulletin/MS03-007.mspx`, March 2003.

[15] Microsoft. LSASS vulnerability. `http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx`, April 2004.

[16] J. Oikarinen and D. Reed. RFC 1459: Internet Relay Chat Protocol, 1993.

[17] Stéphane Racine. Analysis of Internet Relay Chat Usage by DDoS Zombies. Master's thesis, Swiss Federal Institute of Technology Zurich, April 2004.

[18] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[19] The Honeynet Project. Know your enemy: Tracking botnets. `http://www.honeynet.org/papers/bots/`, March 2005.

# An Architecture for Developing Behavioral History

Mark Allman[†], Ethan Blanton[‡], Vern Paxson[†]
[†]*International Computer Science Institute,* [‡]*Purdue University*

**Abstract–** We present an architecture for large-scale sharing of past behavioral patterns about network actors (e.g., hosts or email addresses) in an effort to inform policy decisions about how to treat future interactions. In our system, entities can submit reports of certain observed behavior (particularly attacks) to a distributed database. When deciding whether to provide services to a given actor, users can then consult the database to obtain a global history of the actor's past activity. Three key elements of our system are: *(i)* we do not require a hard-and-fast notion of *identity*, *(ii)* we presume that users make *local* decisions regarding the *reputations* developed by the contributors to the system as the basis of the trust to place in the information, *(iii)* we envision enabling *witnesses* to attest that certain activity was observed *without* requiring the witness to agree as to the behavioral meaning of the activity. We sketch an architecture for such a system that we believe the community could benefit from and collectively build.

## 1 Introduction

A key problem in trying to prevent unwanted traffic is that interactions on the Internet are largely anonymous and self-contained. The host offering services often has no idea who is requesting the service or what sorts of activity the requester has undertaken in the recent past. Lightweight anonymous access to information has been fundamental to the proliferation of network services. However, the Internet's wide-open nature has also fueled the spread of worms, viruses, spam, DDoS attacks, and other forms of unwanted traffic.

Consider some requester, $R$, attempting to access some service, $S$. What does $S$ know about $R$? Several bits of information *may* be available, such as:

- If IPsec [7] or TCP's MD5 option [5] are used then $S$ may have a reasonably solid notion of the *identity* of $R$, and therefore can make a determination of whether to provide access to the given service.
- $S$ may be able to query a local cache of $R$'s previous activity. For instance, a local Intrusion Detection System (IDS) may track behavior exhibited by remote hosts over time.
- $S$ may have access to information about $R$ collected and shared by remote entities. For instance, [11] outlines a scheme for one instance of Bro to receive event notifications over the network from other Bro's. Additionally, systems such as *www.dshield.org* aggregate simple activity reports from a vast number of locations to form a picture of a host's recent activity. Another class of mechanisms proactively test hosts to determine some characteristic, which is then publicly disclosed (e.g., $R$ is an open SMTP relay).
- $S$ could check $R$ against a list of host "types". For instance, lists exist that identify DSL, cable modem or dial-up IP addresses (e.g., *www.sorbs.net*), and a number of spam-fighting systems block or score email based on whether a remote host is on such a list.

The above systems all provide useful information that can help form policy. However, the information is quite often narrow in scope and/or difficult to obtain — especially when trying to thwart unwanted traffic in real-time. For instance, locally-collected IDS information may be quite rich, but it only relates to hosts that attempt to access the local network. Further, setting up pair-wise trust between IDS systems is an arduous process when done at any scale. Likewise, setting up and managing pair-wise keys for IPsec or MD5 can be burdensome and contrary to the nature of providing publicly available services. Finally, databases containing the "type" of IP address offer a broad view of the network, but offer no indication of the behavior of individual hosts.

Our goal is to overcome these limitations. We envision doing so by devising a system that can accumulate reports of unwanted traffic in a general fashion across the entire network. As mentioned above, there are some systems currently providing instances of such services. However, given their centralized nature these lack the robustness we would desire in a system called upon to absorb large numbers of queries and provide actionable information that we can use in real-time. In addition, the consumers of the data provided by these services must both trust the information and agree with the policies used to aggregate the reports and derive the information. Finally, we believe an opportunity exists to consider a *general* architecture for tracking the behavior of large numbers of actors, where both "behavior" and "actor"

can encompass a wide range (as discussed below).

In this paper we elucidate an architecture for a *distributed* system that provides a lightweight actor-based history database. Rather than requiring a large series of pair-wise key exchanges, a user of the database makes *local* decisions regarding the degree to which to trust information in the database, primarily in terms of the user's local assessment of the submitter's *reputation*. While we primarily illustrate the architecture in terms of tracking purported attacks associated with Internet hosts, the architecture generalizes to tracking general types of purported *behavior* (e.g., attacks) associated with *actors* (e.g., Internet hosts or IP addresses).

The system we propose does not require aggregation of information in the database into a succinct "status" of a given actor. Rather, the database provides information to consumers who make their own determination as to the validity and trustworthiness of the information, and then formulate a local policy decision based on these assessments.

By having judgments of reputations and reports remain local decisions, we aim to address some of the issues with how users can possibly trust such a wide-open database that will inevitably include misinformation. An additional, and arguably more powerful, mechanism for developing such trust is the inclusion in the system of *witness* reports: these are additional records that corroborate part of the *context* of other reports. For example, a router might corroborate that a certain packet passed its way, without needing to agree that the packet constituted an attack. Because the router is attesting to a low-level fact rather than a high-level judgment, the hope is that well-known parties (e.g., ISPs) will find it affordable in terms of both processing and reputation to contribute such reports.

This paper presents an architectural overview of the system, not a detailed analysis of any particular facet. Much more work must be done by the community to bring such a system to fruition. Our hope is to refine the thinking behind the system by receiving community input. We present an overview of the information in the database in § 2; discuss the role of policy in § 3; analyze attacks on the system in § 4; present additional issues in § 5; and sketch future work in § 6.

## 2 Database Overview

We maintain the history database in a Distributed Hash Table (DHT) [3] designed to support querying in massively distributed environments (e.g., PIER [6]). DHTs provide a robust and distributed system for storing information, meaning that each DHT node need only cache a fraction of the information in the database, while complete information is efficiently made available to every node. The use of a DHT also minimizes the amount of damage that duplicitous database nodes can cause, while encouraging arbitrary systems to join the database.

Entries in the database are cryptographically signed by the entity which submits them. Each record is inserted under two hash keys: one hash key identifies the actor about whom the entry relates, and the other hash key is a cryptographic public key associated with the submitter. These allow consumers to look up behavior associated with particular actors (e.g., the IP address of a remote host attempting to access one of the consumer's services) and the set of all records submitted under a given key, respectively.

The cryptographic keys used in the system need not be widely known or authenticated in any way; they serve only to correlate records in the database by tying them to a particular submitter. For our purposes, it is often enough to know that two records were submitted by the same entity without necessarily having a clear idea of who that entity is. The use of cryptographic signatures allows entities which retrieve a set of records to have some degree of confidence that the records were inserted by the same entity (or at least inserted under collusion), as well as opening the possibility for local policy decisions to "trust" certain foreign keys. We discuss this and other opportunities for using keys to influence policy in § 3.

We store several kinds of information in the hash table. The basic record is a *behavior record*: a statement that the reporting entity believes it observed a given type of behavior perpetrated by a given actor (e.g., that a given IP address launched a given attack). Records may also be inserted by *witnesses* that vouch that certain events occurred, but not whether these events reflected the purported behavior. For example, a witness could report that they observed a particular packet, without needing to agree that it constituted an attack. Such records form *audit trails* that can lend credibility to reports of particular behaviors. Finally, entities which make use of a particular record in the database may note that they have done so, and thus make it known that they have some level of trust in the record. Or they might note that they declined to make use of a particular record. In either case, they might later record their post facto assessment of their own decision.

### 2.1 Behavior: Attacks

In this section we illustrate the general framework for tracking behavior using attacks from Internet hosts as a specific example. The reader should however keep in mind that the scope of "actor" and "behavior" could in principle be considerably broader. For example, the domain of actors might be email From addresses, with the corresponding behaviors reflecting the use of the addresses in phishing or spamming email. As an example that focuses on behavior but not attacks, we might

imagine a system for which actors are blogger Web page URLs, with the associated reports being submitters' assessments of the accuracy and utility of the postings.

For tracking attack behavior, when a participating host or edge network (e.g., an IDS) determines that "unwanted" traffic has arrived they may indicate this by inserting a corresponding record in the distributed database. They would insert the record twice, once using the IP address (i.e., actor's identity) of the purported malicious host as the hash key and once with the reporter's public key as the hash key. Each record might consist of the following fields:

- *Timestamp*: The time when the first instance of the behavior was received. The DHT node that stores the record also records a *report timestamp* indicating when it received the report. As discussed in § 4, the relative time reports are made could have implications in terms of the reputation of the reporter (and, therefore, independent timestamps may be useful).
- *Actor identity*: A characteristic of the actor creating the behavior that can be used to retrieve reports of that actor's behavior. For some types of behavior, the identity to associate with them may appear obvious. For example, for observations of network attacks, the IP address of the purportedly malicious host will often make sense. Clearly, equating identity with an IP address is not quite right. However, IP addresses can be acted upon in enforcing security policy. Also, future namespaces (e.g., HIP [9]) could be readily incorporated into the architecture.

  In other cases, the notion might be more diffuse, and might even involve *multiple* identities. For example, testing email for possible spam might be done in terms of looking up each of the relaying servers listed in "Received:" headers. Thus, we may find it useful to think in terms of "identity components" or "identity hints", rather than a single notion of identity.
- *Protocol* and *port number*: The reporter could optionally report the protocol and port numbers on which the unwanted traffic arrives. In some cases this could be useful, for example when subsequent traffic from the given host is likely benign (e.g., web requests from a host with an open SMTP relay are not necessarily likely to be problematic). In other cases, an attack may span a wide variety of ports and/or protocols (e.g., a port scan), in which case this information is difficult to express and has little utility.
- *Behavior observed*: a code roughly describing the unwanted traffic observed. We envision encoding the behavior in broad categories rather than trying to highlight specifics. For instance, a reporter of attacks would indicate "worm", not "Code Red". Possible attack categories might be: worm, virus, spam, scanner, DDoS attack, etc.
- *Behavior digest*: This field contains a fingerprint of the specific activity observed. For an attack, it might be a packet digest [10] of the packet that triggered the report. The point of capturing a behavior digest is to correlate the record with audit trails (as discussed below).
- *Signature*: the cryptographic signature of the above fields and the associated public key.

Defining when to place something in the database is behavior-dependent. For instance, if an IDS finds the signature for a well known attack (e.g., Slammer) then it could record an event immediately. However, an *SSH* connection that attempts to log in with a bad username one time may not be cause for recording an event (given that someone might have made a mistake or a typo). Or, this might warrant a delay in the inserting of the event in the database to better ascertain whether the *SSH* service is under attack.

### 2.2 Witnesses

To add credence to a reporter's database entry we allow for the insertion of *witness statements* in the database. These records do not indicate particular behavior, rather they indicate that the witness indeed observed some element that is purported to be part of the behavior. One possible source of witnesses would be in the routers that forwarded the unwanted traffic to the reporter. If such routers kept records of the traffic traversing their networks in digest form (e.g., for traceback [10] or for reporting packet obituaries [2]), the reporter could request these systems to insert witness statements into the database (signed by the witness' public key).

The statements form the basis of an *audit trail* that at least supports the reporter's contention that a particular traffic stream arrived. In keeping with the locally-determined nature of decision-making in our system, the precise use of this audit trail in assessing a reporter's reputation or setting policy is left for each database user to determine.

### 2.3 Signatories

A final item for the database to store is annotations of records based on their use (or not) in setting policy. In its simplest form, sites or hosts that make use of a record in forming a decision to treat some traffic in a non-standard way (blocking, rate-limiting, heightened surveillance) can sign that record to indicate that they used the information to form policy. These hosts are indicating that they have used this record as part of their policy in how to treat an actor without actually observing specific behavior (which would cause an independent behavior report

to be entered). The signatory mechanism is designed to play a part in determining the reputation of reporters by recording which of these reports other sites have found useful.

The decision to use a particular attack record in the database, or to sign and report its usage, does not need to be solely based on the information from the database. It could be partially based on local policy and observation, as well. For instance, a report of an IP address as a scanner by some arbitrary reporter may not be believed on its own. However, if an incoming packet from that address is destined for a dark portion of a site's address space, then these two pieces of information together may be enough to enact a policy of blocking the given IP address. In this case, the given record in the database should be signed to indicate that it played a part in designing a particular policy.

Finally, we envision two possible refinements to the use of signatories. The first is the reporting of negative use; i.e., that a site decided not to use a reporter's report about a given actor. This allows other sites to more quickly learn of the skepticism of peers they deem trustworthy. The second is *follow-up*, in which sites later report whether subsequent activity led them to a different conclusion than they formed initially. Follow-up is basically a form of report revocation, per § 5.

## 3 Policy

The point of the database is to inform local security decisions with a broader context than that which can be locally collected. The database does not determine that a given actor is somehow malicious (or, more generally, exhibited a particular behavior). Rather, it simply stores and provides reports from sites that have made such determinations. The reports from the distributed database can be combined with any other information on-hand (e.g., IDS records or topology information) to make policy decisions. In using such reports a key component is *trust*. The consumers of the reports have to somehow trust that the report providers inserted valid records in the database.

Database records could be invalid for three reasons: (*i*) the record provider makes inaccurate assessments of an actor's behavior, (*ii*) the record provider is intentionally inserting inaccurate records in an attempt to use the database as the basis of a denial-of-service attack on an actor (or, more generally, to *frame* an actor), or (*iii*) the information contained in the record was accurate at one time but is now out-of-date. To address the first two validity issues we depend on a reporter's *locally-determined reputation* to assess the degree to which the information will be trusted. The third issue is addressed by aging information in the database.

As discussed in § 2, each entry in the database is signed by the reporter. We do not impose a tight coupling between the identity of the reporter and the key used to sign records; rather the history of the key's reports can be used to assess a reporter's aptitude in determining malicious behavior. Reputation is a *local* calculation based on various pieces of information culled from the database. See, for example, [1] for an illustration of one particular scheme for using a distributed database to assess the reputation of participants in a peer-to-peer system. We leave a concrete definition of reputation as future work (and, indeed, there can be various calculations depending on what a particular site values); however, we note several aspects that could be taken into account when assessing the fidelity of reporter $X$:

- The number of distinct reporters (and their reputations) making the same assessment as $X$. This lends weight to the fact that $X$ is not inaccurate or malicious.
- The number of signatories on reports made by $X$. This indicates how many others have some amount of trust in the given report (perhaps due to corroborating evidence, as discussed in § 2).
- An audit trail of the events that $X$ has reported. While the third-party statements that form the audit trail do not confirm $X$'s assessment of some property of the given actor, they do offer evidence that $X$ did in fact observe a traffic stream with the given digest (with a high probability). This can be used as evidence that $X$ may be acting honestly.
- Local evidence that concurs with $X$'s findings; for instance, if $X$ identifies some host $H$ as a scanner and local information has not yet made that determination but has noted several attempts by $H$ to access dark IP addresses.
- Reputations can be earned within different contexts. For instance, a particular $X$ may have a highly accurate worm detector, while having a sub-par scan detector that is often wrong.

Each of the above bases for reputation can be gamed by a malicious reporter. The algorithms for deriving a reporter's reputation therefore need to be tuned to be skeptical of a reporter until the reporter has a track record that is supported by evidence and previously well-known reporters. See § 5 regarding the problem of bootstrapping such a system.

While assessment of reputation can be used to take care of some aspects of invalid information in the database, it cannot account for out-dated information. In fact, out-dated information may well lead to the reputation of a reporter suffering because a once-valid report is now seen as invalid. Therefore, we need some way to age out old information. If we retain the information in the database and allow users to apply their own aging poli-

cies, this would likely yield a database full of information that consumers always essentially factor out, wasting resources. Therefore, we likely need the DHT nodes to prune "old" information, either deleting it or perhaps using sketch-like schemes [8] to aggregate older information in a form of graceful degradation.

## 4 Cheating

A key component of the system sketched above is that it does not require the identity of the actors to be known, but rather depends on the track record of various entities. This leaves the door open for abuse from malicious actors. One concern in this regard is misbehavior by the nodes participating in the DHT, to prevent legitimate information from being inserted into the database or from reaching users in response to queries. In general, DHTs are susceptible to a variety of attacks [4], and resisting these (for example, by restricting membership in the DHT) is an area of active research that is not unique to our needs. Thus, in this section we focus on a second class of cheating, namely attackers inserting bogus information in the database that is, itself, an attack (e.g., noting that one's enemy has a worm in the hopes that this will cause denial of service to said enemy).

In previous work, [1] shows that a workable reputation system can be built such that reporters providing a good amount of bogus information can be identified as cheaters. However, that work also shows that catching reporters that generate only a small amount of misinformation is difficult. Therefore, additional reputation assessment techniques are likely needed, based on the sorts of misinformation that could be placed into the database in our system. We consider the four kinds of misinformation an attacker could place into the database.

First, a malicious user could attempt to place bogus reports in the database in the hopes of denying service to the given hosts. For dealing with some simple versions of such attacks, see [1]. However, a more motivated attacker could retrieve database entries for a key, $K$, that represents a solid contributor (which could be determined by looking at signatories or by running some known, widely-used reputation algorithms) and then insert attack records with the same information as those submitted by $K$ but signed with the attacker's key, $K'$. If the attacker monitors the database for a period of time and continues to report entries based on those of $K$ (or, even a few keys to mask the blatant copying) then $K'$ can steal solid reputation built by $K$, and at this point $K'$ can start inserting bogus records.

One way to resist such attacks is to incorporate the time when an attack was reported, with the first reporter getting more "reputation points" than subsequent reporters. For this defense to work, it is critical that the database correctly record reporting times. In addition, the reputation assessment may highlight "unique reports" to ferret out reports that only $K'$ makes that others do not (though this can be countered with multiple malicious keys colluding). More speculatively, an honest, high-reputation reporter could occasionally insert "ringers" into the database: false records that are revealed as such after a delay. Any other reporter who also reported the ringer is unmasked at that point as having copied the original report.

A second form of misinformation is hosts filing false witness statements in an attempt to bolster their false attack claims. This might be mitigated by asking ISPs to sign all their witness reports with well-known keys.

Third, hosts could add fake signatories to falsified attack reports in an effort to add weight to those reports. If the signer of some record engaged in the above "reputation stealing" scheme then it is conceivable that hosts could be fooled into impeding an innocent actor's traffic. The reputation assessment could take into account the set of signatories in an effort to see if they overlap with known solid keys, or not.

Finally, attackers could flood the database with useless records, to render legitimate use of the database computationally infeasible, to greatly complicate reputation calculations, or even for purposes of spamming advertisements if a mechanism exists that will display entries in the database to users (e.g., for error handling). This threat is quite significant because the system purposely does not tie reporter keys to specific entities. Therefore, it lacks the means to prevent an attacker from minting an endless stream of keys to avoid having a single key identified as a flooder.

It may be possible to diminish the problem of flooding by requiring that reports be corroborated by witnesses that themselves have established reputation in a more global manner (perhaps by independent auditing, as discussed below in § 5). Clearly, there is much future work to do in terms of developing reputation assessment techniques that can ferret out or resist complex patterns of misbehavior.

## 5 Other Issues

In this section we tackle a variety of topics that require additional work as the community moves from an architectural concept to building a real system.

**Deployment:** The architecture we sketch should be incrementally deployable because users do not require global coverage to gain benefit from the system. However, bootstrapping issues remain, both in terms of starting the system as a whole and then adding reporters later. What sort of bar must a reporter meet before they can be reasonably trusted? How does a new consumer of the information in the database know whom to trust? Is there some sort of calibration system that could be developed

to help new users of the database verify which reporters are providing solid information? The path for a new reporter seems less burdensome because the reporter is just informing the database of local decisions already being made. As the reporter develops a track record, the reported entries gain more usefulness globally. This does not greatly benefit the reporter, but perhaps no additional incentive is needed in this regard — it is annoying to be attacked, and for many victims satisfying by itself to engage in some sort of response, even one without direct benefit.

**Linking Keys and Identity:** A key to the scalability of the system is that keys are linked to reputations (which can be independently computed). However, there are pros and cons associated with keys becoming linked with identity. For instance, if network operator A told network operator B, whom they trusted, how to identify the reports A sent to the database, then B may decide to trust those reports without bothering to compute a reputation for A. (Of course, this can work in the reverse and B could decide that A is sloppy and thus untrustworthy.)

However, if an adversary knows that a particular key reports attacks on a given address space, then they could avoid that address space to avoid being reported. (E.g., it is easy enough to check the database after performing a scan to see if the attack triggered an entry, and if so, which key was used to sign it.) If attackers avoid networks that report malicious activity, this can create an incentive for network operators to report malicious activity in the database. On the other hand, in a global sense this could work against trying to quickly contain malicious hosts.

Additionally, if an attacker determines that some address space is linked to a particular public key in the database, the user could use this information in an attempt to determine the site's security policy. For instance, a malicious user may be able to determine what sort of criteria the site uses to declare that a host is a scanner. Such knowledge may let the user explore the site (from additional compromised hosts, say) without triggering alarms.

Finally, if a key is known to represent some organization and the organization's reports to the database are often wrong, then this could cause embarrassment to the organization (and, hence, be a disincentive for participating).

**Revoking Reports:** Another aspect of the system that requires thought is how (or whether) a given reporter corrects mistakes it has inserted into the database (by annotating, not deleting records). For instance, suppose someone from a given host attempted to *SSH* into some system using an invalid username. This is not an infrequent attack method, and as such it could be reported. However, if this was later determined to be a simple case of a user making a typo in a username, then it seems reasonable for the reporter to indicate that the given report was benign after all. Allowing the database to support this is not difficult, but the question remains of what happens within the reputation assessment. The assessment should probably give the reporter some credit for correcting a mistake. However, at the same time we could view the need to correct as an indication of sloppiness because the maliciousness was not soundly assessed before entering the initial report.

A more speculative application of revocations would be after an infected machine is cleaned up. For instance, if some host was flagged as spewing spam and later closed an open relay, one approach would be for the reporters that noted the spammer to indicate that they now believed the host was no longer sending spam. This path becomes messy because it requires that the reporters have some way to determine a given host has been cleaned up. Alternatively, the architecture could allow for a separate entity (e.g., the victim's ISP) to enter such a record (and of course they would accrue reputation for the fidelity of their cleanup reports).

Finally, the system needs to support key revocation in order to flag a previously-valid key as now compromised and untrustworthy, lest an attacker steal the key's reputation. This could be done by inserting a self-signed revocation in the database as one of the entries under the key's hash.

**System Openness:** The openness of the system is a balancing act. Ideally, we would like to allow only honest participants and to deny anyone with ill-intent. The degree to which the system can remain open may largely hinge on the power of the reputation assessment techniques used and the system's ability to resist database flooding attacks.

As discussed thus far, we have considered the system to be open for use by anyone. However, there are advantages to restricting membership in the DHT itself to only known parties to reduce the trustworthiness issues involved in the infrastructure that supports the database. Likewise, closing audit reports to a certain subset of participants also seems tractable and useful.

In addition, the entire system could be instantiated multiple times, each scoped in terms of who can report to and/or query from the database. Such a scoping could prove burdensome due to additional credential requirement. On the other hand, the credentials would be known by the DHT and would still likely be easier to manage than the pair-wise interactions currently required for sites to share information. Furthermore, the complexity of the reputation assessment system in such an environment, while not being completely eliminated, would likely be substantially reduced (e.g., because the likelihood of "reputation stealing" attacks would be quite

small). Finally, we note that for some environments, such as the Grid, our system could leverage an existing identity infrastructure.

**Overhead:** The system outlined in this paper has a cost in terms of the network resources used. The worst case is that each session establishment initiated by a remote host would require a database query to assess the possible intent of the requester. Even a modest-sized institute such as ICSI (a few hundred hosts and a dozen public servers) is visited by many thousands of remote hosts each day, and this figure could become much larger if an attacker were spoofing IP source addresses, though we could offset this by requiring that connections first establish before then deferring their further progress until we can vet their source. We could also gain benefit by caching the lookups from the database, but must do so carefully to avoid using stale information in the face of a fast-spreading attack.

**Surrogates:** A final avenue for investigation is the use of *surrogates* to aggregate the information from the database such that local resources are not required for this task. Clearly, this requires that a consumer trust the surrogate to accurately aggregate information from the database in a timely manner; in particular, to precompute reputations of reporters (as opposed to summarizing per-actor behavior). Such trust could be established based on external relationships. For instance, a company's field offices could trust a surrogate at headquarters to do the computational work of determining reporter reputations nightly and then distributing the information. More speculatively public web sites could be setup to serve information about reputations (or even their judgment of given actors) based on the information culled from the database. In this case, trust goes back to the track record of the surrogate's reports. Also, given that the information being aggregated is generally available, we can audit aggregation sites periodically to verify their trustworthiness.

As a practical matter it may be worthwhile to store snapshots of the database periodically in "data warehouses". Whereas our belief is that information about the distant past is not useful in terms of current malicious activity, such a warehouse would allow us to test this hypothesis. In addition, new reputation algorithms and circumvention attacks based on the database's timeouts can be analyzed.

## 6 Future Work

We have sketched the beginnings of an architecture to store and retrieve activity reports. Here we focussed on the big picture, but to bring such a system to fruition will require community efforts on a number of fronts: (*i*) developing workable locally-computable reputation algorithms, (*ii*) obtaining acceptable performance in terms of lookup overhead, timeliness of information propagation, and resilience to flooding, and (*iii*) devising workable witness digests and associated audit trails, as a key element in providing a low barrier-to-entry when developing a reputation.

## Acknowledgments

Many thanks to Joe Hellerstein, Petros Maniatis, Scott Shenker and the anonymous reviewers for their thoughtful comments. This work was supported in part by the National Science Foundation under grants ITR/ANI-0205519, NSF-0433702 and STI-0334088, for which we are grateful.

## References

[1] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, 2001.

[2] K. Argyraki, P. Maniatis, D. Cheriton, and S. Shenker. Providing Packet Obituaries. In *Proceedings of ACM SIGCOMM HotNets-III*, 2004.

[3] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking Up Data in P2P Systems. *Communications of the ACM*, 46(2):43–48, Feb. 2003.

[4] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Security for Peer-to-Peer Routing Overlays. In *Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Dec. 2002.

[5] A. Heffernan. Protection of BGP Sessions via the TCP MD5 Signature Option, Aug. 1998. RFC 2385.

[6] R. Huebsch, J. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of the* $29^{th}$ *VLDB Conference*, 2003.

[7] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol, Nov. 1998. RFC 2401.

[8] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based Change Detection: Methods, Evaluation, and Applications. In *ACM SIGCOMM Internet Measurement Conference*, Oct. 2003.

[9] R. Moskowitz and P. Nikander. Host Identity Protocol Architecture, Jan. 2004. Internet-Draft draft-ietf-hip-arch-02.txt (work in progress).

[10] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Single-IP Packet Traceback. *IEEE/ACM Transactions on Networking*, 10(6):721–734, Dec. 2002.

[11] R. Sommer and V. Paxson. Exploiting Independent State For Network Intrusion Detection. Technical Report TUM-I0420, Technische Universitat Munchen, Nov. 2004.

# The Spoofer Project: Inferring the Extent of Source Address Filtering on the Internet

Robert Beverly
*MIT CSAIL*
rbeverly@mit.edu

Steven Bauer
*MIT CSAIL*
bauer@mit.edu

## Abstract

Forging, or "spoofing," the source addresses of IP packets provides malicious parties with anonymity and novel attack vectors. Spoofing-based attacks complicate network operator's defense techniques; tracing spoofing remains a difficult and largely manual process. More sophisticated next generation distributed denial of service (DDoS) attacks may test filtering policies and adaptively attempt to forge source addresses. To understand the current state of network filtering, this paper presents an Internet-wide active measurement spoofing project. Clients in our study attempt to send carefully crafted UDP packets designed to infer filtering policies. When filtering of valid packets is in place we determine the filtering granularity by performing adjacent netblock scanning. Our results are the first to quantify the extent and nature of filtering and the ability to spoof on the Internet. We find that approximately one-quarter of the observed addresses, netblocks and autonomous systems (AS) permit full or partial spoofing. Projecting this number to the entire Internet, an approximation we show is reasonable, yields over 360 million addresses and 4,600 ASes from which spoofing is possible. Our findings suggest that a large portion of the Internet is vulnerable to spoofing and concerted attacks employing spoofing remain a serious concern.

## 1 Introduction

The Internet architecture provides no explicit mechanism to prevent packets with forged headers from traversing the network. Malicious parties can leverage the ability to forge or "spoof" the source address of IP packets to mount various attacks. For example, the vulnerability of BGP routers to predictive, spoofed TCP resets was a major concern in the last year [5]. By spoofing the source address, an attacker or compromised host can send packets toward a victim anonymously. This anonymity greatly complicates the job of network operators trying to defend their networks. More insidiously, attackers with the ability to spoof can leverage reflectors [15], making distributed denial-of-service (DDoS) attacks particularly problematic.

Previous research investigates various means of mitigating spoofing. Jin et. al give a scheme to block spoofed packets based on hop count [11]. Bellovin proposes a probabilistic marking scheme to trace spoofed packets to their origin [2], while Snoeren suggests an efficient hash-based traceback mechanism [17]. Despite these research efforts, finding the source of spoofed packets remains an operationally difficult problem for network operators [7].

Techniques such as ingress and egress address filtering [6] and unicast reverse path forwarding checks [1] are employed in some production networks to prevent spoofing. However, current filtering practices are limited by multi-homing, route asymmetry, filter list maintenance and router design. Thus, filtering is generally practical only at the edge of the network and is not universally applied.

Is spoofing still a relevant issue? The rise of zombie farms, where spoofing provides little additional anonymity for an attacker, suggests spoofing may not be a useful technique. The proliferation of Network Address Translation (NAT) devices renders spoofing attacks from hosts behind the NAT useless as the IP header is rewritten. Despite these two factors, analysis of backscatter [13, 14] shows spoofing is still widespread.

Moreover, the difficulty in defending against spoofed attacks suggests that next generation attack farms may intelligently probe the network and adaptively change behavior based on the ability to spoof. Consider a 10,000 node zombie DDoS attack [3]. Assuming a worst case scenario where zombies are widely distributed, a network operator must defend against attack packets from 5% of the routeable netblocks. However, if 2500 of those zombies are attached to networks permitting spoofing, a significant volume of the traffic would appear to come

from any of the routeable netblocks. Not only would the attack be difficult to track and filter, spoofing complicates mitigation of the remaining non-spoofed traffic.

This paper presents an Internet-wide active measurement spoofing project. Clients distributed around the world attempt to test various filtering policies by sending a series of spoofed UDP packets. In contrast to backscatter analysis which observes only the result of spoofing and is oblivious to the identity of the true sources, our research is concerned with finding the portions of the network that allow spoofing. Our results are the first to quantify the extent and nature of source address filtering and the ability to spoof on the Internet. Our key findings are:

1. Approximately 24% of the observed netblocks, corresponding to 25% of the observed autonomous systems, allow spoofing.
2. Filtering is frequently applied inconsistently allowing for partial spoofing of portions of the IP address space.
3. No geographic region in our sample appears to be a predominate potential source of spoofed packets.
4. In over 36% of our cases, filtering policies are applied exactly on the routing boundaries observed in a global BGP routing table.

## 2 Methodology

All major operating systems require administrative privileges to send arbitrarily crafted packets as there are few legitimate reasons for spoofing. There is no way to coerce remote Internet hosts, to which we have no access, to send spoofed packets. Our testing therefore requires the cooperation of willing participants. We make publicly available a "spoofer" program, in source and binary formats. This program allows users to test their own network and records the results on our server. The software as well as continually updated summary statistics are provided on the Spoofer Project home page: `http://spoofer.csail.mit.edu`.

While our coverage is limited by the number of hosts which run the spoofer, we receive test reports from a non-trivial portion of the Internet. Section 3 examines coverage in detail.

### 2.1 Spoofer Operation

As depicted in Figure 1, in a spoofing test the client attempts to send a series of spoofed UDP packets to a server on our campus (step 1). As our server receives the packets, they are recorded in a database for later retrieval

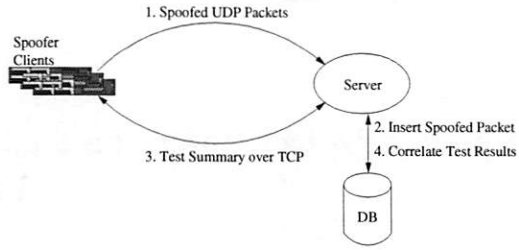



Figure 1: Spoofer test operation. Clients source a series of spoofed UDP packets to our server where they are subsequently disambiguated and analyzed.

Table 1: Source addresses of spoofed packets

| Spoofed Source | Description |
|---|---|
| 1.2.3.4 | Unallocated |
| 172.16.1.100 | Private (RFC1918) |
| 6.1.2.3 | Valid (In BGP table) |
| $IP \oplus (2^N)$ for $0 \leq N \leq 24$ | Neighbor Spoof |

(step 2). Because UDP does not guarantee reliable delivery, the tester sends five packets for each source address with a random inter-packet delay between $(0, 500]$ms to prevent incorrect inferences due to loss.

After sending the spoofed UDP packets, the program establishes a TCP connection with the server to exchange test results and complete the client's test run (step 3). To avoid any secondary filtering effects that might drop test or report packets for reasons other than source address filtering, we use a UDP destination port of 53 for the spoofed packets and TCP port 80 for the test summary exchange. These ports correspond to the well-known DNS and HTTP ports respectively and should be open in the majority of circumstances[1].

The source addresses, summarized in Table 1, are specially chosen to test and infer common filtering policies. The first source address tested is as yet unallocated by the Internet Assigned Number Authority (IANA) [10, 8]. This address should not appear in any routing tables since IANA has not delegated it to any organization. Some networks employ filters that block traffic originating from these unallocated regions of the IP address space.

The second source IP address the program tests is one in a private netblock designated by RFC1918 [16]. Private IP addresses are often legitimately assigned to hosts in a private network for site-local routing, but should never appear on the public Internet. Best common practices dictate that privately addressed packets should be contained within AS boundaries.

Next, the spoofer sends packets with a valid, allocated

source address but one that is spoofed. In contrast to the first series of packets, this address appears in the global BGP routing table but is allocated to another organization. Notably, a filtering policy that allows only packets with source IP addresses that are present in the BGP table is easier for providers to implement and maintain.

Finally, the spoofer attempts to discover the granularity of any applied filtering by successively spoofing addresses from netblocks adjacent to its own. We accomplish this "neighbor spoof" by trying successively larger boundaries until spoofing an adjacent /8. To generate the address, we negate (i.e. flip) a bit at a time in the host's true source address beginning with the least significant bit. Thus, we start by spoofing an adjacent /31 address which corresponds to the host's address $\pm 1$, i.e. the immediate neighbor's address.

For each source address, the spoofer includes a unique random 14 byte identifier string in the payload of the UDP datagram. The unique string allows us to later disambiguate which spoofed packets are received and which are blocked.

During the TCP connection, the client informs the server of its operating system and the number of spoofed sources it sent. For each source, the client reports the identifiers of the spoofed packets it attempted to send. Based on this transaction, the server records in the database the success or failure of the remote machine's ability to send spoofed packets (step 4). Finally, the spoofer runs a traceroute to our server which is also recorded in the database to determine the complete forward path of any spoofs.

At the end of the report transaction, the user is provided a unique URL pointing to a web page containing her test results. This web page summarizes the spoofing and filtering along the path from the user to our host.

## 2.2 Spoofer Run

For each client running the spoofer, several outcomes are possible. The spoofed UDP packets simply may not reach our server if a network filter or other policy blocks them. During the test summary transaction phase (step 3) of the spoofer, the server determines which packets the client believes it sent actually did not arrive.

In some cases the operating system may not allow sending spoofed packets even when running with administrative privileges. The most common instance of this restriction is Windows XP with Service Pack 2 (SP2)[2], however we see other operating systems with security restrictions that cause the client to fail. Alternatively, the packets may arrive with the true source address instead of the spoofed address if a NAT device rewrites the header.

Finally, some or all of the spoofed packets may arrive. In this case, the server will correlate the message identifiers the client reports with the identifiers of the UDP packets it received. In all cases, the server records the outcome in the database for later analysis.

# 3 Results

We advertised availability of the spoofer application on the North American Network Operators Group (NANOG) and dshield security mailing lists. Between March and May 2005, we received 570 client reports, 459 of which were unique [3]. All of our results exclude any data from machines belonging to our own campus netblocks and we count multiple reports from the same client IP addresses only once.

While the general user population may not be interested or motivated to run our spoofing test application, filtering policies are applied to netblocks and hence consistent over the size of that netblock. Because network routing advertisements are aggregated, the granularity of our netblock view may not exactly reflect filtering policies. Whether the netblocks coincide with globally advertised routing prefixes is a question we consider in Section 3.4. Therefore, for the purposes of coverage, it suffices if a single user is able to test a netblock. Unless there is direct evidence of the ability to spoof we classify netblocks as "believed unspoofable."

## 3.1 Failed Spoofs

In this subsection, we consider only the three types of spoofed addresses described in the methodology (unallocated, private and valid) and ignore neighbor spoofing for the time being. As previously discussed, all spoofing for a given client may fail for a variety of reasons. We do not infer anything about the netblocks of clients with these failures:

- *Socket creation blocked by Windows XP SP2:* 122 of the 216 reports from Windows (56%) indicate that the spoofer application was unable to create the raw socket and send spoofed packets. The only version of Windows which blocks packets with a spoofed source address is Windows XP running Service Pack 2 (SP2). That over half of the Windows machines are blocked implies that SP2 is widely adopted in our client population. Because of the dominant popularity of Windows as a client operating system [4], continued uptake of SP2 will have a significant positive impact on preventing spoofing from compromised hosts.

- *Socket creation blocked by other operating systems:* An additional 20 clients not running Windows were unable to send spoofed packets even when running

Table 2: Observed spoofing coverage

| Metric | Spoofable | Believed Unspoofable |
|---|---|---|
| Netblocks | 73 | 229 |
| IP Addresses | 21.0M | 70.0M |
| ASes | 52 | 150 |

Table 3: Spoofing coverage relative to observed and routeable space

| Metric | Spoofable (% Observed) | Spoofable (% Routeable Space) |
|---|---|---|
| Netblocks | 24.2% | 0.04% |
| IP Addresses | 23.0% | 1.31% |
| ASes | 25.7% | 0.29% |

with root privileges. These machines likely employ additional security mechanisms such as capabilities or had local packet firewalls that block the raw socket creation or sendto system calls.

- *Hosts behind NAT devices:* When the UDP packets arrive at our server but with a source address other than the source address they are sent with, the server marks them as rewritten by a NAT. We count all instances of NAT rewriting as a failed spoof since we cannot infer whether or not the host would have been successful had the NAT not been in place. 110 of the clients are behind a NAT device.

- *Totals:* 284 clients, or approximately two-thirds, failed to spoof any packets.

## 3.2 Spoofing Coverage

We quantify the coverage of spoofing along several dimensions. Using RouteViews [12] data, we determine the netblock and AS of each spoofer client. Based on the size of the netblock, we can determine the number of IP addresses the report approximately represents. Table 2 gives the number of netblocks, addresses and ASes that are spoofable and believed unspoofable as observed in our data set.

At the time of writing, there are approximately 1.59B globally routeable IP address, 18,000 autonomous systems and 169,000 netblocks. Table 3 presents the spoofing coverage relative to the addresses, ASes and netblocks we observed and the total global counts.

Approximately 24% of the observed netblocks, corresponding to 25% of the observed autonomous systems, allow some form of spoofing. Assuming a uniform distribution of testing, projecting these numbers to the entire Internet yields over 360M spoofable addresses and over 4,600 spoofable ASes. Our findings suggest that a large portion of the Internet is still vulnerable to spoofing, implying that concerted spoofing attacks remain a serious concern.

Table 4: Frequency of inconsistent filtering. Check marks indicate the presence of a particular filter.

| Private | Unallocated | Valid | Instances |
|---|---|---|---|
| ✓ | ✓ | ✓ | 229 |
| ✓ | ✓ | | 21 |
| ✓ | | ✓ | 0 |
| ✓ | | | 52 |
| | ✓ | ✓ | 0 |
| | ✓ | | 0 |
| | | ✓ | 0 |

## 3.3 Inconsistent Filtering

Many hosts experience inconsistent filtering where a subset of the spoofed packets arrive at our measurement station. We classify the different subsets of spoofing into these inconsistent filtering categories. Again, for the moment we ignore adjacent netblock spoofing. Table 4 summarizes the inconsistent filtering behavior we observe. A check mark in the table indicates the presence of a particular type of filtering, whereas no check mark indicates the absence of filtering.

- *Failed on private address (RFC1918), other spoofs successful:* 52 clients are able to spoof both valid and unallocated addresses, but are unable to spoof private addresses. Since blocking private addresses is an easy and time-invariant policy, it is unsurprising to find instances of only RFC1918 ("martian") blocking.

- *Failed on private and unallocated addresses, valid (routeable) spoof successful:* 21 clients could not spoof the RFC1918 or the unallocated ("bogon") addresses, but could spoof the valid address. This interesting class of inconsistency likely arises from policy that checks for a valid routing entry before forwarding packets. An advantage to providers in adopting this policy is that there is no need to continually and manually update packet filters as new addresses are allocated. An example of a community-wide project to provide a BGP feed of bogon routes to automate filtering is the Cymru bogon route-server project [18].

- *Failed on unallocated addresses, other spoofs successful:* We saw no inconsistencies of this type, implying that providers who are conscientious enough to block unallocated addresses also block RFC1918 packets.

## 3.4 Understanding Filtering Granularity

Next we turn to the problem of understanding filtering granularity. To determine the filtering granularity, we analyze the data from the adjacent netblock neighbor scan. Recall that the client attempts to spoof successive netblocks adjacent to its own, beginning with its immediate neighbor (i.e. a /31) and ending by testing an adjacent /8.

Figure 2 displays the granularity of either ingress or egress filtering employed by service providers tested in our study. If the filtering occurs on a /8 boundary for instance, a client within that network is able to spoof 16,777,215 other addresses. In our study nearly 40% of clients are able to spoof addresses upto a /8 netblock.

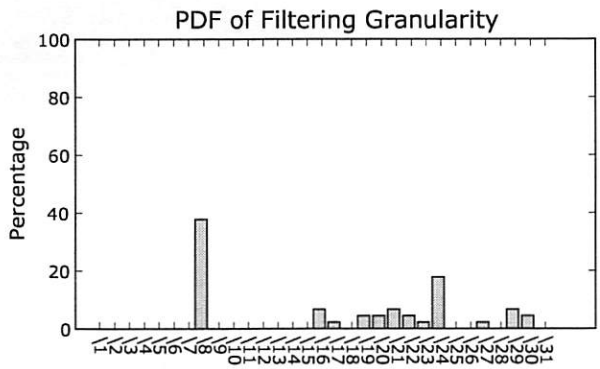



Figure 2: Probability distribution of filtering policy granularity as inferred by neighbor spoof scanning

We are also interested in how closely a host's inferred filtering boundaries match the size of its prefix in the global BGP routing table. This is important because our ability to infer the scope of spoofing is limited if filtering is applied at a very granular level, but routing announcements are highly aggregated. More generally understanding the correspondence between policy enforcement points and routing advertisements as seen from the global routing tables is useful knowledge in that inferences can be made about a provider's network structure and operational practices.

We define a "prefix distance" metric as the difference between the size of the advertised BGP prefix to which the client belongs and the maximum-sized adjacent neighbor netblock the client successfully spoofs. In cases where the client is able to spoof our other three test source addresses, the client is able to also spoof all possible neighbors. We ignore clients that are capable of full spoofing when analyzing filtering boundary effects.

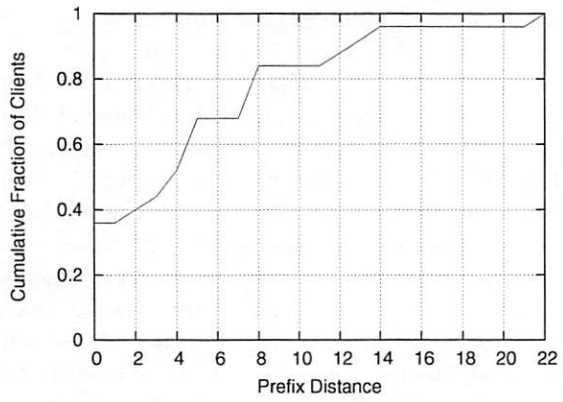
Figure 3(a) plots the cumulative distribution of clients as a function of prefix distance. Figure 3(b) examines the difference between the advertised and inferred address counts. Over 36% of the clients infer the same filtering prefix boundary as the actual route advertisement.

A final curious class of inconsistencies for which we have no good intuition is due to hosts that are able to spoof one of our sources, but are unable to spoof their immediate neighbor's address. In total we found two instances of non-neighbor spoofability. Although there are several possible explanations for this phenomenon, it is unexpected.

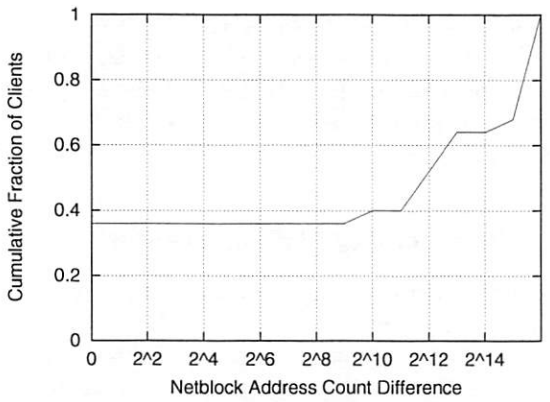
## 3.5 Visualizing the Extent of Spoofing

When our measurement host receives a spoofed packet from a client, we can definitively say that no device along the path from the client to our server performed any filtering to block the packet. To visualize the scope, boundary and geographic extent of spoofing, we use CAIDA's Otter tool [9]. The coordinate space places nodes with a radius corresponding to their AS path length depth from our measurement host and a degree representing geographic location of the AS. The images plot our measurement host in the center of the graph at depth zero labeled as "MIT(AS3)." For each client, we determine the AS path from the client to our measurement host and add each AS as a node in the graph. ASes are connected by edges according to the client's route toward the measurement host. The degree of each node is determined by the longitude of the organization responsible for the AS. In this fashion, we can visualize the geographic position of ASes as well as their BGP distance from the server.

Figure 4(a) contains all ASes of nodes from which we received test reports as well as the intermediate ASes on the path to our measurement host. We see several distinct clusters of nodes corresponding to Europe, Asia and North America. Figure 4(b) plots the ASes and paths corresponding to clients that successfully sent spoofed packets. The ASes are in the same coordinate system so that we can compare the two graphs and determine the boundary and range of spoofing as observed from our host. Comparing the two plots, no geographic area emerges with significantly differing relative ability to spoof. While the second graph is more sparse, a significant portion of the graph is spoofable. In general if filtering is not applied by the providers at the "edges," spoofed packets travel unabated across the Internet. Any filtering that is in place on the three major peering sessions our campus maintains had no effect in preventing spoofed packets from reaching our server.

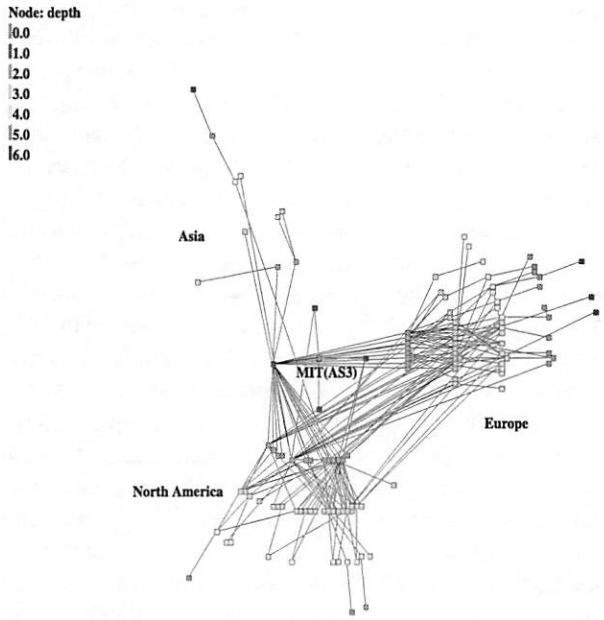



(a) Cumulative distribution of difference between advertised and inferred prefix size.

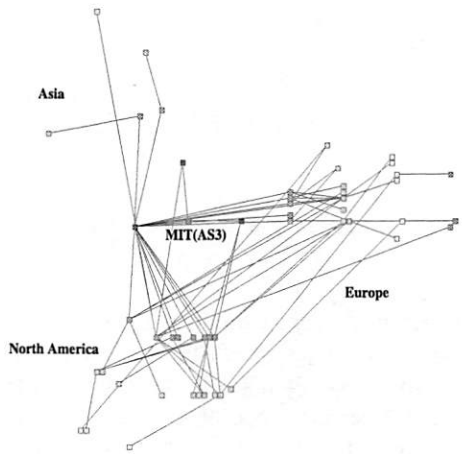



(b) Cumulative distribution of difference between advertised and inferred netblock address counts.

Figure 3: Correspondence between filtering granularity as inferred by neighbor spoof scanning and BGP advertised prefixes in a global routing table.




(a) AS graph of all attempted spoof paths




(b) AS graph of spoofable paths

Figure 4: Visualizing geographic reach of spoofing. Each node represents an AS while edges show routing paths. The AS radius is determined by distance from our measurement host, degree by AS longitude. The graph of spoofable paths is a subset of all paths tested. Our server is in the center of the graph at depth zero.

## 4 Conclusions

To the best of our knowledge, this paper presents the first Internet-wide results examining the extent and nature of source address filtering. Our findings are germane to both network research and operational communities. We see that a significant fraction, approximately one-quarter, of the netblocks, IP addresses and ASes observed permit spoofing. This suggests that a large portion of the Internet is still vulnerable to spoofing and concerted spoofing attacks remain a serious concern. Projecting our results to the entire Internet yields over 360M spoofable addresses and 4,600 ASes from which spoofing is possible. This is particularly significant if next generation attack farms intelligently probe the network and adaptively change behavior based on the ability to spoof.

Currently, we are working on adding functionality to the spoofer that detects what point filtering is applied. In addition, we plan to test spoofing in the reverse direction; the ability to send spoofed packets to random Internet hosts. Combined with continued collection of spoofing reports[4], we hope this project serves as a significant step forward in understanding Internet filtering.

## Acknowledgments

The authors would like to thank Mike Afergan, John Curran, Simson Garfinkel, Aaron Hughes, Ken Shores, Karen Sollins, John Wroclawski and countless NANOG members for constructive discussions and feedback. We also thank the maintainers of the RouteViews project for continuing to provide a valuable community resource.

## References

[1] BAKER, F., AND SAVOLA, P. Ingress Filtering for Multihomed Networks. RFC 3704, Mar. 2004.

[2] BELLOVIN, S. M. ICMP traceback messages. IETF Internet Draft, Sept. 2000. http://www.cs.columbia.edu/~smb/papers/draft-bellovin-itrace-00.txt.

[3] BERINAT, S. Online extortion: How a bookmaker and a whiz kid took on an extortionist and won. *CSO Magazine* (May 2005).

[4] BEVERLY, R. A Robust Classifier for Passive TCP/IP Fingerprinting. In *Proceedings of the 5th Passive and Active Measurement (PAM) Workshop* (2004), pp. 158–167.

[5] DALAL, M. Improving TCP's robustness to blind in-window attacks. IETF Internet Draft, May 2005. http://www.ietf.org/internet-drafts/draft-ietf-tcpm-tcpsecure-03.txt.

[6] FERGUSON, P., AND SENIE, D. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, May 2000.

[7] GREENE, B. R., MORROW, C., AND GEMBERLING, B. W. ISP security: Real world techniques. NANOG 23, Oct. 2001. http://www.nanog.org/mtg-0110/greene.html.

[8] HUBBARD, K., KOSTERS, M., CONRAD, D., KARRENBERG, D., AND POSTEL, J. Internet Registry IP Allocation Guidelines. RFC 2050, Nov. 1996.

[9] HUFFAKER, B., NEMETH, E., AND K. CLAFFY. Otter: A general-purpose network visualization tool. In *Proceedings of INET* (June 1999), pp. 22–25.

[10] IANA. Internet Assigned Number Authority IP address allocations. http://www.iana.org.

[11] JIN, C., WANG, H., AND SHIN, K. Hop-count filtering: An effective defense against spoofed DoS traffic. In *Proceedings of the 10th ACM International Conference on Computer and Communications Security (CCS)* (2003), pp. 30–41.

[12] MEYER, D. University of Oregon RouteViews. http://www.routeviews.org.

[13] MOORE, D., VOELKER, G. M., AND SAVAGE, S. Inferring internet Denial-of-Service activity. In *USENIX Security Symposium* (2001), pp. 9–22.

[14] PANG, R., YEGNESWARAN, V., BARFORD, P., AND PAXSON, V. Characteristics of Internet Background Radiation. In *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference* (Oct. 2004).

[15] PAXSON, V. An analysis of using reflectors for distributed denial-of-service attacks. *ACM Computer Communications Review (CCR) 31*, 3 (2001).

[16] REKHTER, Y., MOSKOWITZ, B., KARRENBERG, D., DE GROOT, G. J., AND LEAR, E. Address Allocation for Private Internets. RFC 1918, Feb. 1996.

[17] SNOEREN, A. C., PARTRIDGE, C., SANCHEQ, L. A., JONES, C. E., TCHAKOUNTIO, F., KENT, S. T., AND STRAYER, W. T. Hash-based IP traceback. In *Proceedings of ACM SIGCOMM* (2001).

[18] THOMAS, R. Team cymru bogon route-server project. http://www.cymru.com/.

## Notes

[1] If these well-known, well-used ports fail or are proxied, attempts to spoof almost assuredly fail as well under most reasonable network configurations.

[2] http://www.microsoft.com/technet/prodtechnol/winxppro/maintain/sp2netwk.mspx

[3] Notably no users or network administrators reported any abuse as a result of running our spoofer.

[4] We continue to receive ~10 reports per day and plan to exploit distributed testbeds for additional coverage.

# Stress Testing Traffic to Infer Its Legitimacy

*Nick Duffield and Balachander Krishnamurthy*
*AT&T Labs–Research, 180 Park Avenue, Florham Park, New Jersey, 07932, USA*
{duffield,bala}@research.att.com

## Abstract

Adaptation in the face of performance degradation is the hallmark of well-behaved network traffic. For sufficiently robust applications, we propose distinguishing good from bad traffic on the basis of the response to artificial performance impairments. We explain the basic requirements for our scheme, and show how it could be generically applied at different levels in the protocol stack.

## 1 Introduction

This paper proposes a new measurement-based methodology for detecting badly behaving network entities. The methodology can potentially be applied across a range of protocol levels. The key idea is to stress test an entity by impairing its network performance, and then to observe the response of the entity to the impairment. Our work is primarily oriented closer to the edge rather than the middle of the network.

The usefulness of the method rests on two assumptions:

- **Differentiation:** The response to impairment is different for bad entities than good entities, and can hence be used to classify entities as good or bad.
- **Recovery:** good entities can recover from impairment in the sense that the performance degradation they suffer remains within acceptable levels.

In practice, these two assumptions are coupled. Suppose we impair the performance of a protocol that is used by an application. We require the underlying protocol to be more sensitive and responsive to impairment that the application, i.e., it must adapt to the impairment before the application suffers an unacceptable performance degradation. Impairment must be sufficiently short-lived to be interpreted as a transient, forcing retry, leading to recovery. The nature of this response is used to distinguish bad from good.

For some applications, e.g., online gaming and others that are highly sensitive to loss and delay, *no* impairment is tolerable. We assume that these applications can be identified (e.g. on the basis of TCP/UDP port numbers) and excluded from our scheme. More generally, the impairments induced by stress testing must not cause performance degradation to exceed acceptable limits, e.g., as specified in a Service Level Agreement (SLA).

## 2 Impairment and Responses

We now describe the broad characteristics of impairments, propose some schemes for good/bad classification, and discuss the costs of misclassification.

### 2.1 Impairment Characteristics

In many cases, applications will experience performance impairments even in the absence of stress testing, for example due to packet loss, delay, network rerouting or resource contention at endpoints. We call these impairments *ambient*. We assume that applications are well-designed in the sense that they a reasonably good at adapting to or otherwise withstanding ambient impairments. If this assumption holds, then it makes sense for impairments introduced by stress testing to conform as closely as possible to the characteristics of the ambient impairments. For then applications need only recover from impairments of the type (even if not the intensity) of those that they are known to adapt to. Another reason for stress impairments from to conform to ambiance is to make it harder for an adversarial bad application to detect that it is being stress tested.

This begs the question of the extent to which ambient impairments can be characterized and then reproduced in stress testing. We will discuss this further for specific

applications in Section 3. Here we enumerate some general parameters for impairment. In our model, the stress testing takes the form of a sequence of individual stress events, characterized by the following parameters:

- *Frequency:* of stress events during stress testing
- *Event duration:* the duration of an single stress event
- *Stress duration:* the duration of a set of stress events. Note that the stress duration may itself be adaptive in the sense that stress testing is terminated once sufficient information has been extracted from responses for classification, or if it is decided that no such classification can be made.
- *Granularity:* at which stress take place. For example, a stress test might target all packets from an IP address, or a subnet, or using a particular UDP/TCP port.

How should these parameters be determined? The aggressiveness of the stress test should be commensurate with the perceived threat. The frequency of bad traffic may be bootstrapped by learning directly from the identification scheme that is used in the stress test, although their will be a transient period in which such estimates are unreliable. One way to seed an initial the threat level is to monitor the intensity of bad traffic taking place, e.g., attacks on "dark" address space as seen in a honeypot [9]. A generic scheme to set parameters for the stress test is described in Section 2.3 below.

## 2.2 Impairment and Control

We use the following three outcome model to describe the response of an entity to a stress event:

- *Cease:* the entity ceases all activity. This may be due to a bad entity having turned its attention elsewhere, or any entity suffering an unrecoverable error as a result of impairment. Whatever the reason, the current stress test of the entity is terminated.
- *Evolve to Good:* adaptation to impairment in the manner expected of a good entity. The current stress test of the entity is terminated.
- *Evolve to Bad:* any other result, for example, not responding to impairment in any manner. The entity is flagged as "suspicious". Stress testing may continue.

How should the outcome of one or several stress events be interpreted? The greater the number of stress events, the greater the certainty of the classification based upon the outcomes. This motivates a cautious approach in which only multiple suspicious outcomes would result in an entity being designated bad, in which case some further action, e.g., blocking, is taken. This may be accomplished in a number of ways; here in one:

- *Fixed horizon classification:* a fixed number $n$ of stress events is generated in sequence. An entity is classified as bad if the number of suspicious outcomes exceeds some level $m \leq n$. The history of outcomes is discarded for each new stress test. probes.

Impairment may also be used as means of control; in this case the intensity of impairments (as determined by frequency and/or duration of stress events) can be increased in response to the level of suspicion. At the same time, it is desirable not to penalize an entity unduly for suspicious behavior far in the past. Thus one penalizes bursts of suspicion, while admitting ambient low level of false positives. This motivates:

- *Queue-based classification:* Suspicion flags are enqueued in an infinite buffer which drains at some rate. The impairment intensity is a function of the buffer occupancy.

## 2.3 Costs of Impairment

We use the term *cost* to denote the potential undesirable impact of the impairment method on applications. We identify two types of cost:

- The **impairment cost** represents the performance degradation experienced by good applications during impairments, and during their response to impairments. This cost includes, for example, the impact of packet loss or delay due to impairment on an application's performance.
- The **identification costs** are those of actions taken on the basis of the identification of applications as good or bad. Examples of such costs include the the false positive rate (the frequency with which good applications are misidentified as bad) and the false negative rate (the frequency with which bad applications are misidentified as good). Impairment may be combined with other classification methods. In this case, the relevant identification costs are those of the composite classifier.

For a number of reasons, both impairment and identification costs may vary with time. As noted, a stress test may adapt to the responses of the target, either to increase the effectiveness of classification, or to control entities classified as bad. Adaptation takes the form of changing

the intensity of the impairment, or possibly removing it altogether. This results in a concomitant changes in the false negative and positive rates.

## 3 Applications

We now examine the key set of protocols on a case by case basis. The main issues are: the possible responses as a result of impairment (apart from normal response and no response), how the responses may be interpreted at the client side, are they recoverable, and if so how (where will the adaptation be seen if at all). Impairment and monitoring has to take place at the same protocol level to facilitate tracking of the reaction. Our guiding principle is that our scheme becomes difficult to use at a given protocol level if there are a variety of adaptation mechanisms possible or used by traffic using that protocol that cannot be distinguished. Differentiation typically requires moving to a higher layer.

Note that in some protocols/layers impairments already being used or subsumed by other mechanisms. Where appropriate we mention them. For example, in BGP the notion of impairment is not so crucial as the peers exchanging routes and/or traffic have prearranged thresholds.

### 3.1 UDP and its Applications

UDP does not provide any adaptation mechanisms in itself. However, higher level applications and protocols may select UDP as their transport, but implement their own adaptation to impairment. In some cases the adaptation to be evident at the transport layer, involving reduction of the sending rate (to avoid congestion) and the repetition of packets (for reliability). The precise details of the adaptation will depend on the higher level protocol or application in question. The UDP port number can be useful in identifying the application. More subtle adaptations may only be visible in the transport payload, and again their location and interpretation would rely on accurate identification of the application in question.

For a specific example, we consider the Realtime Transport Protocol RTP [14] running over UDP. Within RTP, The RTP control protocol (RTCP) provides feedback on the quality of data distribution. Under impairment due to stress testing, receivers will notify senders of reduced quality via RTCP. A good sender is expected to modify its transmissions accordingly. However, the nature may appear quite subtle at the network level , e.g., reducing the number of layers sent from a layered encoding of audio or video. The precise nature of this adaptation is determined according to the policy of the end application.

### 3.2 TCP and Other Congestion Avoiding Transport Protocols

TCP is equipped with congestion avoidance mechanisms: a well-behaved TCP adapts to congestion by decreasing its congestion window before growing it again to explore the available bandwidth. The TCP sender detects congestion when acknowledgements are not forthcoming from the receiver, due to loss and/or delay of packets in transit. Thus a TCP connection should both adapt to impairment, and also recover from it. Stress testing of TCP is accomplished by dropping and/or delaying packets, and observing the connection's response. Impairment must not be so intense as to seriously degrade throughput. A connection which either adapts either too slowly, or not at all, is regarded as bad. Thus, depending on the thresholds for classification. TCP sender that is too aggressive may be regarded as bad.

We give more detail on how the framework might be implemented for impairment by packet loss. We require to be able to monitor and possibly impair both directions of a TCP connection. One way to achieve this is to insert the desired functionality below the TCP layer in a receiving host. Impairment is achieved by selectively dropping packets incoming from a sender. Using the terminology from Section 2, a normal TCP connection is expected to "evolve to good" as it adapts to dropped packets.

A number of approaches to passively characterizing TCP connections have been proposed; see [4, 5, 19]. One aim of this work has is to compare measured characteristics with theoretical baselines. Deviations from these baselines would then be used as triggers for routers to impose limits, e.g. by restricting the bandwidth available to a connection in order to restore a fair distribution of bandwidth. Our aim is somewhat different: to identify and penalize bad connections in advance of their impact on other traffic. [4] proposed three categories: flows that are not TCP friendly, flows that use a disproportionate amount of bandwidth, and flows that are unresponsive to packet loss. The last case matches best with our framework; [4] mentions the idea of introducing packet loss impairments, and observing whether reduction in throughput (if any) conforms to expectations.

A more detailed yardstick against which to measure the sender's response to impairment is an estimate of the sender's congestion window obtained by monitoring the receiver-to-sender ACKs. This has been proposed in [5], which uses the ACKs to drive transitions in a finite state machine (FSM) that represents the sender. State transitions due to sender timeouts may be inferred by monitoring sender-to-receiver retransmissions. Parallel FSMs may be run for different flavors of TCP. Losses occurring between the monitoring point and the sender lead to estimation uncertainties, although the impact can be

detected and corrected for to some degree.

We outline how classification of bad senders might proceed if using the congestion window estimation scheme. A sender is flagged as suspicious if, it sends a packet that would not be allowed by TCP under the current estimate of the senders congestion window. The sender is flagged as bad if it sends multiple suspicious packets, e.g., as determined by one of the classification schemes in Section 2.2. In principle, this classification could be made of any connection, whether impaired or not. The advantage of stress testing is that by active soliciting a response from the sender, it enables the identification of bad senders in advance of circumstances where their bad behavior may be problematic. In this example stress testing helps identify overaggressive TCP senders in times of low network congestion, rather than in congestion periods in which they could have a disproportionate impact on other connections. This is somewhat different to the approach of [4], which proposed characterization of higher rate flows during periods of congestion. Once a sender is classified as bad, some action may be taken against it (e.g. dropping, blocking, or otherwise deprioritizing) at the monitoring point, or by reconfiguring routers in the traffic's path.

We propose to evaluate this approach in future work. Initial evaluations will use controlled TCP-like senders that can be configured to act in good or bad fashion. The existing active TCP inference tool `tbit` [13] can likely be modified for this purpose.

Stress testing is intended for connections that are already established. On the other hand dropping packets at the start of a connection, specifically the SYN and SYN-ACK packets used in the three-way handshake that establishes the connection, should be used sparingly or not at all, since the performance impact of preventing connections establish altogether will be far greater.

Although the operational details may differ, a similar approach to stress testing can be adopted with other congestion avoiding transport protocols, including the Stream Control Transmission Protocol (SCTP) [16] and the Datagram Congestion Control Protocol (DCCP) [7].

Although we have described an example in which monitoring and impairment takes place at a connection endpoint, the same could be accomplished in the middle of the network. (The methods of [5] were originally designed for this context). Here, keeping track of the identification status of potentially a huge number of connections is a challenging. But since their are only limited number of potential states in the classification schemes of Section 2.2, one may use compression schemes such as Bloom filters [1] in order to reduced the space needed. For example, in the fixed horizon classification scheme one may maintain a Bloom filter each number of suspicious outcomes, and store keys of connections in it. The current state is then yielded by the highest level Bloom filter that declare a match on the connection key. However, Bloom filters are subject to false positives, which increases the identification costs.

## 3.3 DNS

A disproportionate amount of DNS traffic consists of UDP exchanges involving port 53; apart from zone transfers there is very little TCP traffic. Traditionally DNS clients are configured to send recursive queries as their resolver libraries often do not follow referrals. Local DNS servers can handle iterative referrals that might come back from authoritative DNS servers. One weakness with impairment in DNS is that many clients may ask multiple name servers in parallel for non-authoritative queries. Authoritative DNS servers however can impair requests by referring the request to non-existent servers to see if the query comes back. Root servers are ill equipped to participate given how busy they are with the already large fraction of illegitimate, albeit non-attack, queries [18] that they receive.

## 3.4 SMTP

The Simple Mail Transfer Protocol has reasonable potential for impairment trials. Although email has the reputation for being delivered instantaneously, the dependence on this has significantly eroded due to the popularity of instant messaging and the rise of email spam. A non-trivial number of email messages are delayed for a variety of reasons and the retry mechanism built into the application (with retrials carried out over several days) ensures that the mail will be eventually delivered.

Our assumption here is that spammers are not likely to retry sending the mail as it would require human intervention. Robots would have to be able to parse the reply and separate simple bounce messages from vacation programs and impairment triggered responses. However, depending on the importance of the mail, the human (non-spammer) sender may retransmit the mail. Each such retransmission dramatically lowers the probability of the sender being a spammer. Many sites maintain a whitelist (good senders), blacklist (bad senders), and graylist (as yet unclassified senders). Retransmitters could be moved from the graylist to the whitelist. The related step of moving non-retransmitters to a blacklist is however trickier as is the problem of dealing with mailing lists.

A serendipitous benefit for users who have been selected for impairment is that they faced a one-time delay but future communications are likely to improve. A spammer trying to subvert the mechanism would have to retransmit a very large number of messages in trying to

game this mechanism with a low probability of guaranteed success.

### 3.5 HTTP

At the HTTP layer, redirection would involve a 3xx level response (redirection class response code such as `307 Temporary Redirect` but to a blank page. Alternately the server could send back a `408 Request Timeout` or a `503 Service Unavailable` along with a `Retry-After` header indicating the number of seconds after which the client should retry the request. Here, this value should be as low as possible; say 1 second. The assumption again is that a user may retry but an attack program is not likely to react the same way. With the Web server under our control, it is possible to monitor the frequency of access and retries as input to further impairment actions. Interestingly, the expected adaptation on the part of the user is to retry whereas in TCP the expectation is the opposite: backing off.

We benefit due to the additional and more detailed semantic information available at the Web server. For example, it is possible to distinguish between normal user requests, and those that emanate from spiders. Thus, a Web server operating under impairment rules, can make judicious choices of URLs whose access trigger impairment guided by its access patterns and popularity.

### 3.6 P2P

Three broad categories of P2P protocols have emerged: the original central index model of Napster, the flooding of neighbors with separation of search and download phases model exemplified by Gnutella and KaZaa, and the most popular tracker process and segmented downloading model of BitTorrent. There are at least two schools of thought: one that presumes virtually all the content exchanged on P2P networks are of dubious legality and the other that views that there is a steadily increasing share of significant amounts of legal content being exchanged. For example, Linux Kernel releases are now routinely copied via BitTorrent.

One impairment technique already prevalent is content pollution: insertion of white noise in audio files or unexpected content in large video files, to frustrate the illegal downloaders [10]. Another prevalent technique–that of choking or throttling connections–is to reduce freeloading and to look for other peers who might be able to share wanted content faster. We consider the latter technique here.

Elimination of freeriding that began in eMule and was extended in BitTorrent revolves around the use of tit-for-tat mechanisms by giving poor service to nodes that do not dedicate enough of their bandwidth to uploading. Here impairment is done by deliberately scheduling poor sharers at the rear end of the queue. Another impairment technique available in protocols like BitTorrent is an *early choke mechanism* whereby nodes who are constantly looking for better connected nodes can drop one of the existing (poor) connections. While this is done for a selfish purpose, it can address concerns about nodes that may pretend to have poor connectivity. However, the false positives will be detrimental to the choked node.

The absence of both local history (between a pair of nodes) and global history (across nodes) leads to false positives (well behaved nodes are mistakenly impaired) and false negatives (nodes that don't share enough receive good treatment). Similar to the TCP case where we argued for trying impairments on long lasting connections only, we believe such impairments should only be tried when there is enough information about the other nodes.

## 4 Relation to Existing Approaches

Variants to our proposed impairment notion have been proposed both in the middle of the network and at the edge in the past. In the TCP arena, the notion of penalty box has been advanced, as a way of restricting bandwidth usage by misbehaving flows [4]. At user-level applications such as Email it is possible to challenge email senders with a simple puzzle to be solved in order to distinguish human senders from programs. It is important to note a key distinction between proposals made in the past and ours: the primary issue was fairness in the past with a different quality of service offered to misbehaving flows arising out of, presumably accidental, misconfigurations. We are more interested in identifying malicious and deliberate senders of unwanted traffic. Once an email puzzle is solved the sender is permanently classified as good by being added to a whitelist thus leading to potential abuse by forging addresses of whitelist members.

Honeypots [9, 15], a resource whose value lies in its unauthorized use, advertise unused address space and examine traffic that arrive there. Honeypots can help identify suspicious IP addresses [17]. Some honeypots listen passively, while others actually interact with the traffic by responding to connection set up attempts. At the other extreme, there are honeypots that can emulate a kernel keeping the attacker busy with fake responses. In the past, tarpits [8] have been deployed to actually waste resources of suspicious attack sources. However, the key difference is that virtually all traffic arriving at the dark address space is known to be unwanted. In the domain of electronic mail, the MAPS [11] black list of abusers of email allowed any system administrator to drop all email coming from these domains. Again, the impair-

ment here is after the identification of the (potentially) offensive source.

The throttling of connections suspected not to share an equal fraction of their bandwidth for uploading in P2P networks, is clearly a form of impairment. However, such tit-for-tat mechanisms is on a one-to-one basis and is based on immediate recent history of transactions. Such a mechanism requires evaluation of all of the node's neighbors. The impairment can be temporary as choked connections can be unchoked later on a case-by-case basis. The granularity of impairment is at the connection level rather than on packets. Even the semantic queuing of suspected poor sharers by placing them towards the tail end of the queue can lead to starvation of some nodes indicating absence of fine-grained control.

# 5 Strengths and Limitations

In this section we discuss strengths and limitations of stress testing: the need to control impairment cost, the scope for countermeasures, and the trade-offs in using application level semantics for identification.

## 5.1 Keeping Impairment Costs Acceptable

Impairment costs must not be so large as to make stress testing unacceptable for good network users. Other approaches that have been proposed include doing limited flooding of links that are suspected to be bearing attack traffic [2] and the more recent proposal to attempt to generate challenges to suspected email senders [12]. In the former case there is a risk of doing more damage than warranted or beneficial while in the latter zombie machines generating email may get challenges. We focus on keeping impairment within an SLA or similar limits.

The total impairment experienced by an entity has two components: the ambient impairment and the impairment due to stress testing. The total costs must be kept within acceptable limits. This requires two things.

Firstly, the acceptable limit must be known. This may be expressed in terms of SLAs between a service provider and its customers, that governing the allowable performance degradation and penalties for deviations therefrom. However such limits may need to be applied conservatively, since customers are commonly responsive to degradations beyond an "effective" SLA corresponding to the level of service that they usually receive, which may be better than the worst degradation allowed under the SLA.

Secondly, the level of ambient impairment must be known. Depending on the application, this may be known from application level statistics, e.g. as maintained by Web servers, or from network measurements. In the latter case, one challenge is to characterize ambient impairments at the same level of granularity as the stress test itself. Although commonly reported loss statistics, such as those reported via SNMP, are aggregated at the link level, a more recent method, Trajectory Sampling [3], allows the estimation of loss rate (and sometimes packet delays) at the level of individual hosts and links.

## 5.2 Scope for Countermeasures

A strength of stress testing is that the scope for countermeasures from a bed entity is limited. Firstly, the impairments in a well-designed stress test are not distinguishable from ambient impairments, and thus it is difficult to determine that stress testing is taking place. This likely entails using a full spectrum of likely impairments (e.g. including both loss and latency) in a suitably randomized manner that leaves no signature. Secondly, the method is potentially ubiquitous, making reverse blacklisting by bad entities harder. Thirdly, response to impairment in an aggressive manner makes it more likely that the entity will be flagged as suspicious or even identified as bad.

## 5.3 Application Layer Semantics

In principle a detailed understanding of application semantics may be used to construct detailed models of expected user behavior, and departures therefrom detected. However, this approach becomes more problematic the further up the protocol stack one goes. Applications may be deployed in multiple hosts and in different variants; thus it is necessary to either modify applications or write appropriate dynamically linked libraries. At the network layer, with the small number of transport protocols, it is easier to implement impairments. The trade-off is that with knowledge of application-specification semantics it is easier to construct narrow and specific impairments to identify potential attackers. For example, a Web server could track access patterns on its site and selectively redirect requests for rarely requested URLs when the number of requests exceed a threshold or when they come from BGP prefixes never seen before [6]. However, this requires state maintenance about URL access patterns and tuning when the site changes.

# 6 Impairment factors

Below we list various factors to be kept in mind while considering impairment. They include where impairment should be done, directionality of traffic, for how long, and how meaningful thresholds on false positives and false negatives can be generated.

Impairment is done most readily at the receiving network's ingress point or at the application host. It does not involve any resources in the middle of the network.

Should impairment be done on incoming traffic only or can we consider outgoing traffic too? Depending on the application it may well make sense to impair outgoing traffic. For example, if there is a suspected virus loose behind a network triggering large volumes of email or outbound portscans, it might be reasonable to selectively drop some transactions as a pre-throttling measure. An added benefit in dealing with outgoing traffic is that there is a better prior history of patterns to make informed and selective impairments.

Depending on the infrastructure being protected, the cost of false negatives and false positives, the duration and nature of impairment will vary within applications. A DNS attack against a typical site is a lot less problematic than against a CDN infrastructure which depends on DNS for its entire service. In such instances, cost of a false negative is high and it may pay to be more aggressive. Thus, where there is a potential of amplification of attacks, frequency of impairment attempts can go up.

In some cases, the parameters of the stress test can be set based on a cost analysis related to that in [20]. Suppose that the false positive $f_+$ and false negative $f_-$ rate for identification are known and a function of test parameters $\phi$. This knowledge can come from analysis of a model of the test, or from empirical observations, or both. Suppose a proportion $p$ of the total traffic is believed to be bad traffic. $p$ may be a current estimate based in the stress testing itself, or as determined by independent measurements. Then the parameters $\phi$ are chosen so as to minimize the total cost $C(\phi) = pf_-(\phi) + (1-p)f_+(\phi)$. Note that the test parameters automatically adapt to changes in the measured or expected value of $p$.

## 7 Summary

We have outlined a new technique, impairment via stress testing, to distinguish good and bad traffic. The technique is low cost, tailored to different applicable layers, and thresholdable to ensure no service level agreements are violated. The natural concerns of false positives and negatives are discussed via tunable parameters governing the stress causing events. The technique is more natural at the transport layer than others although we can take advantage of the broader leeway available in SMTP, HTTP, and P2P applications. A key advantage of stress tests is that attackers cannot easily detect them and will likely become more visible by trying to counter the measure.

## Acknowledgments

We would like to thank the anonymous referees for their useful feedback and pointers to earlier related work.

## References

[1] BRODER, A., AND MITZENMACHER, M. Network applications of bloom filters: A survey. In *Proc. Allerton Conference* (Monticello, IL, October 2002).

[2] BURCH, H., AND CHESWICK, B. Tracing anonymous packets to their approximate source. In *Proceedings of Usenix LISA conference* (2000).

[3] DUFFIELD, N., AND GROSSGLAUSER, M. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking 9*, 3 (June 2001), 280–292.

[4] FLOYD, S., AND FALL, K. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking* (August 1999).

[5] JAISWAL, S., IANNACCONE, G., DIOT, C., KUROSE, J., AND TOWSLEY, D. Inferring tcp connection characteristics through passive measurements. In *Proceedings of IEEE INFOCOM* (March 2004).

[6] JUNG, J., KRISHNAMURTHY, B., AND RABINOVICH, M. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *WWW* (MAY 2002).

[7] KOHLER, E., HANDLEY, M., AND FLOYD, S. Datagram Congestion Control Protocol (DCCP). Internet Draft, March 2005.

[8] Hackbusters - Homepage. `http://hackbusters.net`.

[9] LANCE SPITZNER. Honeypots: Definitions and Value of Honeypots. `http://www.tracking-hackers.com/papers/honeypots.html`.

[10] LIANG, J., KUMAR, R., XI, Y., AND ROSS, K. Pollution in p2p file sharing systems. In *IEEE Infocom* (2005).

[11] The maps realtime blackhole list. `http://mail-abuse.org/rbl/`.

[12] NELSON, M. Fairuce, March 2005. `http://www.alphaworks.ibm.com/tech/fairuce`.

[13] PADHYE, J., AND FLOYD, S. On Inferring TCP Behavior. In *ACM SIGCOMM* (2001), pp. 287–298.

[14] SCHULZRINNE, H., CASNER, S., FREDERICK, R., AND JACOBSON, V. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[15] SHAHEEM MOTLEKAR. Honeypots: Frequently Asked Questions. `http://www.tracking-hackers.com/misc/faq.html`.

[16] STEWART, R., XIE, Q., MORNEAULT, K., SHARP, C., SCHWARZBAUER, H., TAYLOR, T., RYTINA, I., KALLA, M., ZHANG, L., AND PAXSON, V. Stream control transmission protocol. RFC 2960, October 2000.

[17] VINOD YEGNESWARAN AND PAUL BARFORD AND SOMESH JHA. Global Intrusion Detection in the DOMINO Overlay System. In *Proceedings of ISOC 2004* (February 2004). `http://www.cs.uwisc.edu/~barford/isoc04.ps`.

[18] WESSELS, D., AND FOMENKOV, M. Wow, that's a lot of packets. In *Proceedings of Active and Passive Measurement Workshop* (April 2003). `http://moat.nlanr.net/PAM2003/PAM2003papers/3836.pdf`.

[19] ZHANG, Y., BRESLAU, L., PAXSON, V., AND SHENKER, S. The characteristics and origins of internet flow rates. In *ACM Sigcomm* (August 2002).

[20] ZOU, C., DUFFIELD, N., TOWSLEY, D., AND GONG, W. Adaptive defense against various network attacks. SRUTI 2005 Workshop (Steps to Reducing Unwanted Traffic on the Internet), Cambridge, MA, July, 2005.

# Adaptive Defense Against Various Network Attacks

Cliff C. Zou[1], Nick Duffield[2], Don Towsley[1], Weibo Gong[1]
[1] *University of Massachusetts, Amherst, MA*
[2] *AT&T Labs Research, Florham Park, NJ*

## Abstract

In defending against various network attacks, such as Distributed Denial-of-Service (DDoS) attacks or worm attacks, a defense system needs to deal with various network conditions and dynamically changing attacks. In this paper, we introduce an "adaptive defense" principle based on cost minimization — a defense system adaptively adjusts its configurations according to the network condition and attack severity in order to minimize the combined cost introduced by false positives (misidentify normal traffic as attack) and false negatives (misidentify attack traffic as normal) at any time. In this way, the adaptive defense system generates fewer false alarms in normal situations (or under light attacks) with relaxed defense configurations, while protecting a network or a server more vigorously under severe attacks. Specifically, we present detailed adaptive defense system designs for defending against two major network attacks: SYN flood DDoS attack and Internet worm infection. The adaptive defense is a high-level system design that can be built on top of various non-adaptive detection and filtering algorithms, which makes it applicable for a wide range of security defenses.

## 1 Introduction

The current Internet is constantly under network attacks. Many defense methods and systems have been proposed to deal with these attacks. These systems typically first detect the on-going attack traffic, then block (filter) the attack traffic accordingly. Attack detection is of crucial importance in such defense systems. An imperfect detection algorithm will inevitably generate detection errors in terms of "false positives" and "false negatives". A "false positive" means incorrectly identifying a normal packet (or connection, or host, etc) as an attack whereas a "false negative" means incorrectly identifying an attack as a normal one.

Most research has focused on stationary network operation with fixed configurations. However in reality, attack detection systems have to face rapidly changing network conditions and various attack intensities. Therefore, besides finding a good detection algorithm, it is equally or more important to design an "intelligent" defense system that can automatically adjust its detection and filtering parameters to achieve the best performance possible under every possible attack situation.

We introduce an "*adaptive defense principle*" based on "cost minimization" — a defense system adaptively adjusts its configurations according to network conditions and "*attack severity*" in order to minimize the combined cost introduced by false positives and false negatives at any time. We call such a defense system as an "adaptive defense system". Compared to a traditional non-adaptive defense system, an adaptive defense system generates fewer false alarms in normal situations (or under light attacks) while protecting a network or a server more vigorously under severe attacks.

Denote by $\kappa_t$ the "attack severity" at time $t$, which can be the fraction or volume of attack traffic, or other metrics determined by the types of attacks. Denote by $\theta_t$ the set of configuration parameters used in the detection algorithm. A defense system's false positive cost and false negative cost at time $t$, denoted by $C_p(\kappa_t, \theta_t)$ and $C_n(\kappa_t, \theta_t)$ respectively, are functions of $\kappa_t$ and $\theta_t$. Whenever the attack severity $\kappa_t$ changes, the adaptive defense system will choose the up-to-date optimal configurations $\theta_t$ by minimizing the combined cost:

$$f = \min_{\theta_t}\{C_p(\kappa_t, \theta_t) + C_n(\kappa_t, \theta_t)\} \qquad (1)$$

We present concrete adaptive defense systems for defending against two major network attacks: SYN flood DDoS attack, and Internet worm infection. The adaptive defense is a high-level system design that can be built on top of various *non-adaptive* detection and filtering algorithms, which makes it applicable for a wide range of security defenses.

The rest of the paper is organized as follows. Section 2 surveys related work. We present the system design for defending against DDoS attack and Internet worm infection in Section 3 and Section 4, respectively. In Section 5 we evaluate the performance of these two adaptive defense systems. Finally Section 6 concludes this paper.

## 2 Related Work

Mirkovic *et al.* [10] presented a comprehensive taxonomy of DDoS attack and defense mechanisms. Many DDoS detection approaches, such as the "IP traceback" [12], or the "MULTOPS" [2], try to find the identities of the real attacking sources. Hussain *et al.* [6] presented a framework to classify DDoS attacks into single-source and multi-source attacks. However, these methods cannot be used directly to block attack DDoS traffic. In order to detect *and* filter SYN flood packets at the victim end, Kim *et al.* [8] provided a general anomaly detection framework. Jin *et al.* [3] provided a concrete "Hop-Count Filtering" algorithm to filter out spoofed attack SYN packets based on packets' TTL values.

For Internet worm defense, Williamson [16] proposed a rate-limiting "throttling" method to constrain infection traffic. "EarlyBird" in [13] and "Autograph" in [7] detect and block worm spreading through identifying the common bit-strings among all infection network traffic of a worm. To prevent internal infection, Staniford [14] presented the segmentation idea to separate an enterprise network into many isolated subnetworks. Jung *et al.* [4][5] presented "Threshold Random Walk (TRW)" detection algorithms to detect and block worm infection based on the excessive number of unsuccessful scans sent by a worm. Weaver *et al.* [15] presented a simplified version of TRW algorithm that is suitable for both hardware and software implementation. Pang *et al.* [11] provided a comprehensive study of the characteristics of the abnormal traffic in the Internet.

Lee *et al.* [9] considered various cost factors, including false positive/negative cost, in the process of developing Intrusion Detection System (IDS). However, such a cost-sensitive design is a static system design method, which does not consider how to dynamically adjust an IDS's configurations according to the attack condition. Our previous paper [17] only briefly mentioned the adaptive defense principle, but never explored it.

## 3 Adaptive Defense System I: SYN Flood DDoS Attack

"SYN flood" attack is a denial-of-service attack by sending a large amount of SYN packets to a network or a server [10]. The attack packets usually have spoofed source addresses to hide the real attacking sources and also make defense much harder. For simplicity, we refer to the victim of a SYN flood attack as a server.

### 3.1 Underlying detection algorithm: extended "Hop-Count Filtering"

The "Hop-Count Filtering" (HCF) algorithm presented in [3] is a concrete and promising approach for SYN flood DDoS attack. In a nutshell, HCF infers the hop-length of a connection request source to a server based on the Time-to-Live (TTL) value in the incoming SYN packet IP header, then compares this value with the real hop-length of the client, which is derived from the client's previous successful connections. If these two values are different, HCF determines that the incoming SYN packet is a spoofed attack packet. Since attackers do not know the real hop-counts from their spoofed source addresses to the victim, spoofing the initial TTL values cannot help attack packets to avoid HCF detection as proved in [3]. Due to space constraint, please refer to [3] for the detail of the HCF detection.

Denote the "*false positive probability*" as $P_p$, the probability of incorrectly dropping a normal packet (or a normal connection, or a normal host for other types of attacks); denote the "*false negative probability*" as $P_n$, the probability of incorrectly treating an attack as a normal one. Due to memory constraint and Internet routing path changes, the HCF can change its detection strictness by allowing curtain deviation of the observed hop-count value from the value saved in its hop-count table. We run the HCF detection on the simulated normal SYN traffic and spoofed SYN flood traffic, respectively. From the simulation, we derive the detection performance in terms of $P_p$ and $P_n$ under different detection configurations. Fig. 1 shows the detection performance trade-off for the server "net.yahoo.com", whose hop-count data is provided to us by authors in [3]. Nine small circles in the figure from the left to the right represent the detection performance under different configurations. We define a detection sensitivity parameter $\delta$ ($0 \leq \delta \leq 8$): each small circle in Fig. 1 from the left to the right corresponds to $\delta = 0$ to $\delta = 8$, respectively.

We extend this discrete HCF to a continuous HCF based on "probabilistic dropping". Suppose the continuous HCF uses a real number $\delta$ as its detection parameter, $0 \leq \delta \leq 8$. Denote an integer $m = \lfloor \delta \rfloor$ and a real value $q = \delta - m$. Then, this continuous HCF will accept all packets acceptable by the discrete HCF with $\delta_1 = m$ while drop all packets that should be dropped by the discrete HCF with $\delta_2 = m + 1$. For the remaining packets that should be dropped by the $\delta_1$ HCF but accepted by the $\delta_2$ HCF, the continuous HCF accepts them with the probability $q$. In this continuous HCF, both $P_p$ and $P_n$

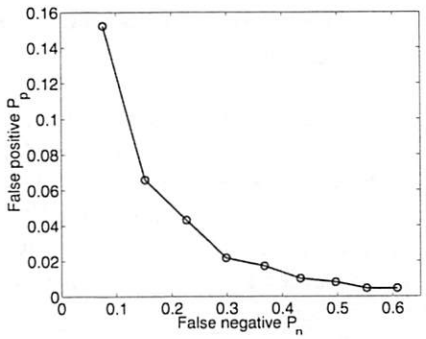



Figure 1: HCF detection performance under different configurations

are piece-wise linear functions of $\delta$. Therefore, $P_p$ is also a piece-wise linear function of $P_n$ as shown in Fig. 1.

## 3.2 System design based on a general cost function

Denote by $\pi$ the fraction of attack packets among all incoming SYN packets. $\pi$ naturally exhibits the relative attack intensity compared to the normal payload of a server, and hence, we use $\pi$ to represent the "attack severity" of a SYN flood DDoS attack.

The adaptive defense system updates its HCF detection parameter $\delta$ periodically at each discrete time denoted by $k$ ($k = 1, 2, \ldots$). During the time interval from $k$ to $k+1$, the HCF implements $\delta(k)$, which corresponds to the pair of $P_p(k)$ and $P_n(k)$. During this time period, the fraction of incoming packets identified by the defense system as attack packets is denoted by $\pi'(k)$, while the real attack fraction is denoted by $\pi(k)$.

$\pi(k)$ differs from the observed value $\pi'(k)$ because: (1) the limited samples within a discrete time interval introduce an observation statistical error; and (2) some attack packets are not counted in $\pi'(k)$ due to false negatives whereas some normal packets are counted in $\pi'(k)$ due to false positives.

In the following we derive an unbiased estimate of the real attack severity, denoted by $\hat{\pi}(k)$. Suppose during the time interval from $k$ to $k+1$, the attack severity $\pi(k)$ does not change and the defense system receives $N(k)$ SYN packets. Then, $\pi(k)N(k)$ packets are attack packets while the remaining $[1-\pi(k)]N(k)$ are normal ones. The defense system drops $\pi'(k)N(k)$ packets, among which $[1 - P_n(k)]\pi(k)N(k)$ are real attack packets and the remaining $P_p(k)[1 - \pi(k)]N(k)$ are falsely dropped normal packets. Therefore, we have

$$\pi'(k)N(k) = [1-P_n(k)]\pi(k)N(k)+P_p(k)[1-\pi(k)]N(k)$$

Removing $N(k)$ from both sides yields

$$\pi'(k) = [1 - P_n(k)]\pi(k) + P_p(k)[1 - \pi(k)] \quad (2)$$

From (2), we derive the estimation formula of $\pi(k)$ as:

$$\hat{\pi}(k) = \frac{\pi'(k) - P_p(k)}{1 - P_n(k) - P_p(k)} \quad (3)$$

Through statistical analysis, we find $E[\hat{\pi}] = \pi$; hence $\hat{\pi}(k)$ is an unbiased estimate.

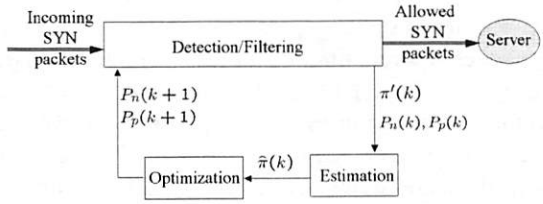



Figure 2: Adaptive defense system architecture

Fig. 2 illustrates the architecture of the adaptive defense system. At the end of time $k$, the adaptive defense system first uses (3) to derive an estimate $\hat{\pi}(k)$ of the real attack severity, then finds the "optimal" detection parameters $P_n(k+1)$, $P_p(k+1)$ (i.e., $\delta(k+1)$) for use in the next time interval. The "optimization" module tries to minimize the combined cost of false positives and false negatives by minimizing:

$$f = \min_{\delta(k+1)} \{c_p[1-\hat{\pi}(k)]P_p(k+1) + c_n\hat{\pi}(k)P_n(k+1)\} \quad (4)$$

where $[1 - \hat{\pi}(k)]P_p(k+1)$ is the fraction of falsely dropped normal packets and $\hat{\pi}(k)P_n(k+1)$ is the fraction of attack packets that pass through to the server.

Those two cost factors, $c_p$ and $c_n$, have concrete physical meanings: they represent the cost of incorrectly dropping (accepting) a normal (attack) SYN packet, respectively. In some cases, they can be chosen as constants whereas in other cases they should be functions of the attack severity. For example, while a server can tolerate a small number of false negatives, beyond some point, the received attack traffic will severely consume the system's resources. Whether to choose constant or functional cost factors should be determined by the specific defense requirement and experiences from security staffs.

## 3.3 "Buffer-aware" performance function

The general cost function (4) is suitable for a wide range of security defense systems. If a server has a specific requirement, however, an object-oriented performance function would achieve a better defense performance.

A server usually has two separate buffers for incoming TCP connections: one for pending connections called

"pending buffer"; another for connections that have been established. The pending buffer is susceptible to SYN flood DDoS attack. Suppose the server's performance is not affected by the number of pending connections in the pending buffer so long as the buffer is not overflowed. Such a server has a specific performance objective: to accept as many as possible normal connection requests.

Define "sojourn time" to be the time period a SYN packet resides in the pending buffer. Denote the average sojourn time of a normal SYN packet as $T_1$ and the average sojourn time of an attack SYN packet as $T_2$. The adaptive defense system updates its parameters periodically at each discrete time $k$ and the time interval is denoted by $\Delta$. The defense system still has the same architecture as shown in Fig. 2. Suppose the server has a pending buffer that can hold $K$ pending TCP connections at the same time. At the end of discrete time $k$, denote the number of incoming packets during the last time interval as $N(k)$.

If the defense system deactivates its filtering functionality and allows all $N(k)$ packets to pass through, the buffer size requirement, denoted by $B^0$, is:

$$B^0 = \frac{T_2}{\Delta}\hat{\pi}(k)N(k) + \frac{T_1}{\Delta}[1-\hat{\pi}(k)]N(k) \quad (5)$$

If the defense system activates its filtering functionality with parameters $P_p(k+1)$, $P_n(k+1)$, then $P_n(k+1)\pi(k)N(k)$ attack packets and $[1-P_p(k+1)][1-\pi(k)]N(k)$ normal packets will pass through the detection/filtering module to reach the server. Thus the buffer size requirement, denoted by $B^1$, is:

$$B^1 = \frac{T_2}{\Delta}P_n(k+1)\hat{\pi}(k)N(k) + \frac{T_1}{\Delta}[1-P_p(k+1)][1-\hat{\pi}(k)]N(k) \quad (6)$$

Therefore, at time $k$, the adaptive defense system should choose its defense parameters $P_p(k+1), P_n(k+1)$ for the next time interval according to:

- If $B^0 < K$, deactivate the filtering functionality (the server's pending buffer will not overflow anyway).
- If $B^0 \geq K$, activate the filtering functionality and choose the optimal $P_p(k+1), P_n(k+1)$ (i.e., $\delta(k+1)$) by minimizing:

$$f = \min_{\delta(k+1)} |B^1 - K| \quad (7)$$

In this way, the server can accept the maximum number of normal SYN packets.

Basically, (7) tries to minimize the cost caused by over-filtering ($B^1 < K$) or under-filtering ($B^1 > K$).

# 4 Adaptive Defense System II: Internet Worm Infection

In this section, we study how to design an adaptive defense system for defending against a fast spreading Internet worm, such as Code Red, Slammer and Blaster [1].

Because a scanning worm blindly scans IP space to find targets, a worm-infected host has a much lower probability to set up successful connections than a benign host. "Threshold Random Walk (TRW)" [4][5] detection is based on the fact that a worm-infected host sends out many more failed connection requests than successful requests. Weaver *et al.* [15] further simplified the TRW algorithm for hardware implementation. We deploy a modified version of the worm detector presented in [15] as the underlying detection algorithm.

Our modified TRW detector works in the following way: each source host that initiates a connection is assigned a non-negative "counter" with the initial value of zero. This counter (if not equals to zero) decreases by one if the source host initiates a successful connection, and increases by one if the source initiates a failed connection. Multiple connection attempts from a source targeting the same destination are treated as one connection attempt (e.g., TCP/SYN retransmission before the timeout). A source host is determined to be infectious when its counter reaches a threshold $W$.

The next step is to represent the Internet worm "attack severity". An enterprise network has a fraction of unused IP addresses. All connection attempts to these addresses, which are called "illegal scans", will always fail. We use the number of illegal scans observed in a discrete time interval, denoted by $Z$, to represent the attack severity.

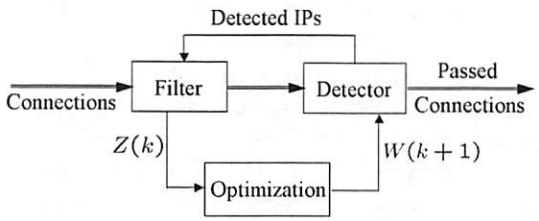



Figure 3: Adaptive defense system architecture for defending against Internet worm infection

Fig. 3 illustrates the architecture of the adaptive defense system. Whenever the "detector" detects an infected host, it sends the host IP to the "filter" where further scanning traffic from the host will be blocked. Denote by $Z(k)$ the number of illegal scans observed from time $k$ to $k+1$ and $W(k)$ the detection parameter used from $k$ to $k+1$. Fig. 3 shows that at the end of time $k$, the system derives the optimal $W(k+1)$ to use for the next time interval based on the current attack severity $Z(k)$.

The "optimization" module derives $W(k+1)$ based on the performance function:

$$f = \min_{W(k+1)} \{c_p \cdot \frac{1}{W(k+1)} + c_n \cdot W(k+1) \cdot Z(k)\} \quad (8)$$

As $W$ increases, an infected host is able to send more scans before it is detected and blocked; but fewer benign hosts would be incorrectly blocked. Therefore, $W(k)$ describes the trade-off: $c_p/W(k+1)$ corresponds to the false positive cost and $c_n \cdot W(k+1) \cdot Z(k)$ corresponds to the false negative cost.

## 5 Evaluation

In this section, we evaluate the defense performance of the above two adaptive defense systems based on either simulation experiments or real attack traces.

### 5.1 Defense against SYN flood DDoS

#### 5.1.1 General cost function

First, we study the adaptive defense system with the general cost function (4). We assume that during each time interval $\Delta$ (e.g., $\delta = 30$ seconds), the server receives 1,000 normal SYN packets. The spoofed SYN flood attack varies its attack intensity as shown in the top graph of Fig. 4. The simulated SYN flood attack includes two types of attack dynamics: (1) attacking traffic gradually increases its intensity (from time 0 to 500); and (2) all distributed attacking hosts begin to send attacking packets at the same time (from time 700 to 800). The bottom graph of Fig. 4 shows how the adaptive defense system automatically tunes its detection parameter $\delta$ (In this experiment, $c_p/c_n = 2$).

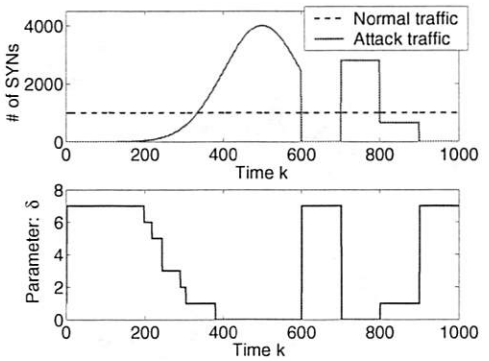



Figure 4: SYN flood attack scenario and the defense system response based on the general cost function (4)

To verify the attack severity estimation (3), we plot the real value $\pi(k)$, the observed value $\pi'(k)$ and the estimated value $\hat{\pi}(k)$ as functions of time $k$ in Fig. 5. This figure clearly shows that Eq. (3) provides accurate estimation results at any time.

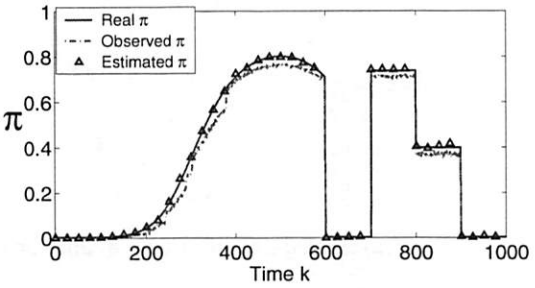



Figure 5: Verification of the estimation formula (3)

#### 5.1.2 Buffer-aware performance function

Next, we study the adaptive defense system based on the buffer-aware function (7). $\Delta = 30$ seconds as used in previous experiment. We assume that normal SYN packets have the average sojourn time $T_1 = 3$ seconds in the pending buffer; attack packets have $T_2 = 25$ seconds (since most attack packets will stay in the buffer until time-out). The pending buffer is assumed to be able to support $K = 200$ connection requests at the same time.

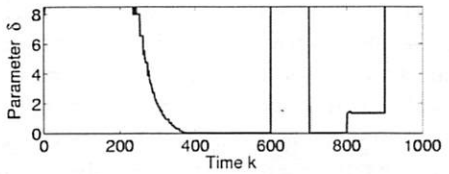
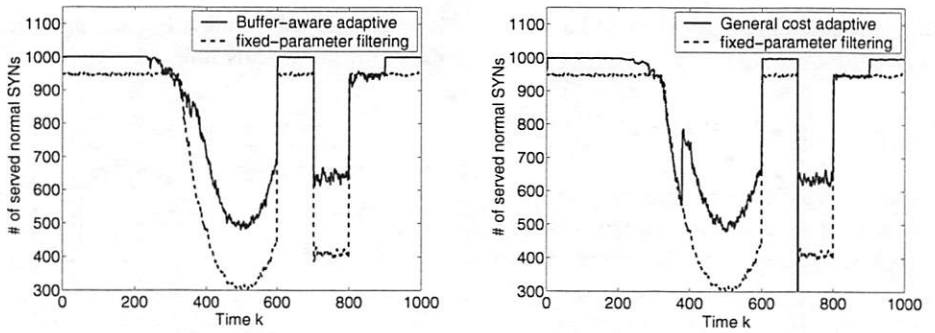



Figure 6: Adaptive defense system response based on the buffer-aware performance function (7)

The SYN flood attack follows the same dynamics as the experiment shown in Fig. 5. Fig. 6 shows how the defense system adjusts its parameter $\delta$. $\delta > 8$ means the defense system deactivates its filtering functionality. This figure and previous Fig. 4 show that both adaptive defense systems have the similar responses. The difference is that the adaptive defense system here has a continuously changing optimal $\delta$ since (7) is a non-linear function of $\delta$ (while (4) is a linear function of $\delta$).

#### 5.1.3 Performance comparison

In the above two experiments, we also obtain the information of accepted normal packets. To see the adaptive defense performance, we also conduct a baseline experiment where a fixed-parameter HCF with $\delta = 1$ is deployed, which is the recommended setting in [3]. Fig. 7(a) and Fig. 7(b) show the defense performance in




(a). Based on buffer-aware function (7) (b). Based on general cost function (4)

Figure 7: Performance of adaptive defense systems compared with the fixed-parameter system

terms of the number of accepted normal SYNs for these two adaptive defense systems, respectively.

Compared with the fixed-parameter defense, both adaptive defense systems accept more normal connection requests either under very light attacks or under heavy attacks. The fixed-parameter defense system uses a set of settings that is optimal only for a specific attack condition, which is not suitable for a real implementation where people expect a defense system to work well under various network conditions.

Fig. 7 also shows that we do not need to design a very accurate adaptive defense system in order to improve the performance of an underlying non-adaptive detection algorithm. As long as we use the adaptive defense principle to adjust a system's settings, the defense performance will be improved more or less. In fact, we run the experiment shown in Fig. 4 many times with different values of $c_p$ and $c_n$, the adaptive defense system always improves its performance compared with the fixed-parameter system in terms of the number of accepted normal requests (similar results as shown in Fig. 7).

## 5.2 Defense against worm infection

In the following experiments, we use a monitored Slammer propagation trace to study the performance of the adaptive defense system. The trace is a tcpdump data containing all UDP packets (targeting at port 1434) received by a /16 network. The top graph of Fig. 8 shows the number of Slammer UDP packets received during each second, which is $Z(k)$ as we use one second for the discrete time interval. The monitored /16 network has two Internet connections. At 150 seconds, one connection went down and caused the monitored Slammer scans dropped suddenly. At 217 seconds, one internal computer was infected and its scanning traffic caused local congestion, and hence, the monitored Slammer scans dropped suddenly for the second time.

The bottom graph of Fig. 8 shows how the defense

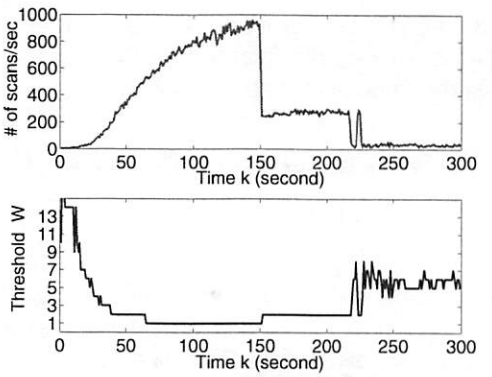



Figure 8: Slammer attack and the response by the adaptive defense system based on TRW detection

system responds to the attack changes by adjusting $W(k)$ (in this experiment, $c_p/c_n = 1000$). $W(k) = 1$ is the most aggressive defense that the system can operate: any host will be blocked (on the suspicious port only) as soon as one illegal scan from it is observed.

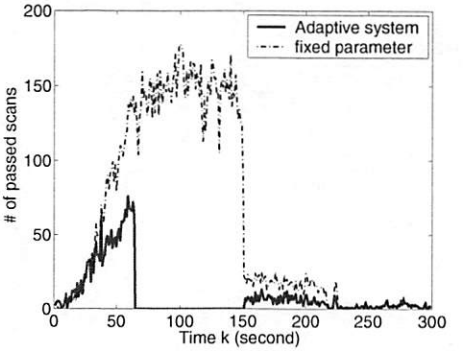



Figure 9: Worm scans passing the defense system

Fig. 9 shows the number of worm scans entering the /16 network — the other worm scans are blocked by the defense system (the peak level of original worm scans is

1000 per second). For comparison, we also show in this figure the case of a fixed-parameter system where $W = 4$. Note that since the number of *vulnerable* computers in a local network is usually much smaller than the number of addresses allocated to the network, only a very small percentage of passed worm scans could possibly cause infection. Of course, if we are very concerned with the worm infection, we can increase the ratio of $c_p/c_n$ to make the defense system quickly updates its threshold $W(k)$ to 1 when $Z(k)$ increases (at the cost of increasing the number of falsely blocked normal hosts).

For the evaluation of false positives, [4] and [15] have used real network traces to show that the "Threshold Random Walk" algorithm has very limited false positives (most of those falsely detected hosts are web crawlers or proxies). Since our adaptive worm defense system uses the similar underlying detection algorithm, we do not repeat such an evaluation here.

## 6 Conclusion

To defend against various network attacks, we introduce an "adaptive defense" principle based on cost minimization — a defense system adaptively adjusts its configurations according to the network condition and attack severity in order to minimize the combined cost introduced by false positives and false negatives at any time. In this paper, we present concrete system designs to defend against two major network attacks: SYN flood DDoS attack and Internet worm infection. The adaptive parameter update includes very simple estimation and optimization, thus the computational overhead is very small. The adaptive defense is a high-level system design that can be built on top of various non-adaptive detection and filtering algorithms, which makes it applicable for a wide range of security defenses.

There are still many work to do on the adaptive defense design. First, we want to further study how to choose the cost factors $c_p$ and $c_n$ quantitatively according to the defense requirements. Second, in order to understand accurately the impact of false positives/negatives, we plan to evaluate the adaptive defense system based on real monitored traces that include both attack and normal traffic. Third, when defense settings are adaptive, attackers might be able to influence the detection in such a way as to deny service to legitimate traffic. We plan to further study this system robustness issue.

## Acknowledgements

We gratefully thank Eric Cronin and Anthony Kurc for sharing their hop-count dataset, and Andrew Daviel from TRIUMF, Canada for sharing his monitored Slammer trace. This work was supported in part by ARO contract DAAD19-01-1-0610, NSF Grant EEC-0313747, EIA-0080119, ANI-0085848 and CNS-0325868.

## References

[1] CERT. CERT/CC advisories. http://www.cert.org/advisories/.

[2] GIL, T. M., AND POLETTO, M. MULTOPS: a data-structure for bandwidth attack detection. In *Proceedings of USENIX Security Symposium* (August 2002).

[3] JIN, C., WANG, H., AND SHIN, K. G. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *Proceedings of 10th ACM Conference on Computer and Communications Security* (October 2003).

[4] JUNG, J., PAXSON, V., BERGER, A. W., AND BALAKRISHNAN, H. Fast portscan detection using sequential hypothesis testing. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2004).

[5] JUNG, J., SCHECHTER, S. E., AND BERGER, A. W. Fast detection of scanning worm infections. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)* (September 2004).

[6] KEROMYTIS, A., MISRA, V., AND RUBENSTEIN, D. A framework for classifying denial of service attacks. In *Proceedings of ACM SIGCOMM* (August 2003).

[7] KIM, H., AND KARP, B. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of 13th USENIX Security Symposium* (August 2004).

[8] KIM, Y., LAU, W., CHUAH, M., AND CHAO, H. Packetscore: Statistical-based overload control against distributed denial-of-service attacks. In *Proceedings of the IEEE INFOCOM* (March 2004).

[9] LEE, W., FAN, W., MILLER, M., STOLFO, S., AND ZADOK, E. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security 10*, 1,2 (2002).

[10] MIRKOVIC, J., AND REIHER, P. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review 34*, 2 (2004).

[11] PANG, R., YEGNESWARAN, V., BARFORD, P., PAXSON, V., AND PETERSON, L. Characteristics of Internet background radiation. In *Proceedings of the Internet Measurement Conference (IMC)* (October 2004).

[12] SAVAGE, S., WETHERALL, D., KARLIN, A., AND ANDERSON, T. Practical network support for IP traceback. In *Proceedings of ACM SIGCOMM* (August 2001).

[13] SINGH, S., ESTAN, C., VARGHESE, G., AND SAVAGE, S. Automated worm fingerprinting. In *Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)* (December 2004).

[14] STANIFORD, S. Containment of scanning worms in enterprise networks. *Journal of Computer Security* (2003).

[15] WEAVER, N., STANIFORD, S., AND PAXSON, V. Very fast containment of scanning worms. In *Proceedings of 13th USENIX Security Symposium* (August 2004).

[16] WILLIAMSON, M. M. Throttling viruses: Restricting propagation to defeat mobile malicious code. In *18th Annual Computer Security Applications Conference* (December 2002).

[17] ZOU, C. C., GONG, W., AND TOWSLEY, D. Worm propagation modeling and analysis under dynamic quarantine defense. In *Proceedings of ACM CCS Workshop on Rapid Malcode (WORM'03)* (October 2003).

# HoneySpam: Honeypots fighting spam at the source

Mauro Andreolini Alessandro Bulgarelli Michele Colajanni Francesca Mazzoni
*Dip. Ingegneria dell'Informazione*
*Università di Modena e Reggio Emilia*
*Modena, Italy, 41100*
{andreolini.mauro, bulgarelli.alessandro, colajanni, mazzoni.francesca}@unimo.it
http://weblab.ing.unimo.it/people/

## Abstract

In this paper, we present the design and implementation of HoneySpam, a fully operating framework that is based on honeypot technologies and is able to address the most common malicious spammer activities. The idea is that of limiting unwanted traffic by fighting spamming at the sources rather than at the receivers, as it is done by the large majority of present proposals and products. The features of HoneySpam include slowdown of the e-mail harvesting process, poisoning of e-mail databases through apparently working addresses, increased spammer traceability through the use of fake open proxies and open relays.

## 1 Introduction

Internet has become the preferred architecture for the most popular user-oriented services. As any popular resource, the Internet-based services are subject to an increasing level of malicious actions which may have bad consequences for both the users and the services providers. In this paper, we focus on the amount of unwanted traffic due to unsolicited e-mails [12, 3], which is widely known as *spam*. Spamming activities often turn into remarkable losses of user time and productivity, and a waste of user and service provider resources.

The most diffused proposals and products for fighting against spam are *receiver-oriented* in the sense that the mail server tends to receive all e-mails and then to determine through some filtering technique(s) whether an e-mail is valid or not (e.g., [19, 16, 7]). Similar techniques can be applied at the client level by the user (e.g., [18, 21, 11]) or can be integrated (e.g., [22]). Independently of the applied techniques that are becoming more sophisticated and valid, these approaches suffer of two main problems: the percentage of low positives is nowadays very low, but there are still false negatives [26, 17]; even worse, receiver-oriented approaches do not reduce the Internet traffic and do not limit the waste of network and disk resources of the service providers.

The alternative idea proposed in this paper is to address the previous issues by working on the *spam sources* instead of the *spam destinations*. A first source-oriented solution can be achieved through filtering mechanisms, such as the use of mail server *black lists* [25]. The destination e-mail server checks whether the IP address of the source e-mail server is inserted into the black list and, in case, denies the receipt of messages. This is a brute force approach with limited effects because an increasing number of spammers use forged SMTP headers in order to hide the real sender. Another source-oriented approach aims to build *white lists* of friendly addresses from which the receipt of spam is highly unlikely; all other sources that are not included in the white lists are not accepted (e.g., [8]). It should be clear that, although more modern white list are dynamically built and self-adaptable, they are really effective just for specific user communities. Apart from theses issues, black and white listing do not help in decreasing the amount of unwanted traffic through the network, but only at the destination.

We follow the idea that fighting against the spam sources cannot be done through one tool, because sending unsolicited e-mails is just the last step of a complex set of operations that are carried out by spammers (see Section 2). Hence, an adequate fight against spam sources requires a framework of different tools and, possibly, the cooperation among interested organizations. The proposal of this paper does not yet include cooperation, but it describes a framework (*HoneySpam*) that is built up of many components among which the pivot is represented by the honeypot technology. The traditional goal of honeypots is to trace the activity of an attacker, locate him through a *traceback* operation, recognize common operational patterns in attacks with the purpose of automatically detecting them in the future. HoneySpam faces the following spamming actions at the source:

- *e-mail harvesting*;
- *anonymous operations*.

E-mail harvesting that is, the collection of valid e-mail addresses through automated crawlers [2, 27], is fought

through emulated Web services. Two different damages are inflicted to crawlers: *crawler slowdown* and *e-mail database poisoning*. First, the crawlers are slowed down into endless loops of Web page retrievals, through the creation of Web pages with many hyperlinks. Second, databases used by spammers to collect the e-mail addresses are polluted. In particular, the pages generated by the Web service contain special, valid, dynamically created e-mail addresses that are handled by emulated SMTP services present in HoneySpam. As soon as the spammer starts to use these freshly collected addresses, it can be traced back and blacklisted.

Fraudulent activities such as spamming are often achieved anonymously, through the use of open proxy chains. HoneySpam provides fake open proxy and open relay services [12, 14] that log every activity and block spam-related traffic.

HoneySpam fights common spamming actions for one organization, although its real potential should be tested in a cooperative context, such as [10].

The rest of the paper is organized as following. Section 2 presents a brief overview of some common spamming activities. Section 3 describes the requirements of an anti-spam tool that emerge from a detailed study of spamming activities and presents the design and some implementation details of the HoneySpam architecture. Section 4 presents an overview of possible countermeasures that the spammers might carry out against HoneySpam and how they are handled. Section 5 discusses related work. Section 6 illustrates future research perspectives. Finally, Section 7 presents some conclusive remarks.

## 2 Spammer activities

In this section, we present a brief summary of the most common actions performed by a spammer.

**E-mail harvesting.** Figure 1 shows one of the first actions performed by spammers that is, *e-mail harvesting*. An automated browsing software (called *crawler*) issues a HTTP request to fetch a Web document (*page 1*). Once the HTTP response is retrieved from the Web server, the crawler parses its content, extracting links to e-mail addresses and to other Web pages. Links to Web pages are used to continue the crawling process (*link 1* to *link k*), while links to e-mail addresses (*address 1* to *address n*) are used to compose lists or (more usually) to populate databases of potential *customers*. E-mail harvesting is an important step, because a spammer has to build its list of victims before sending unsolicited messages to them. It should be pointed out that (thanks to existing collections of e-mail addresses, usually available on CDs or databases), e-mail harvesting is not performed by every spammer, since long lists of (presumably valid) e-mail addresses can be bought from the Internet. Although harvesting actions have softened recently, they still remain one of the major sources of e-mail addresses for spammers [13].

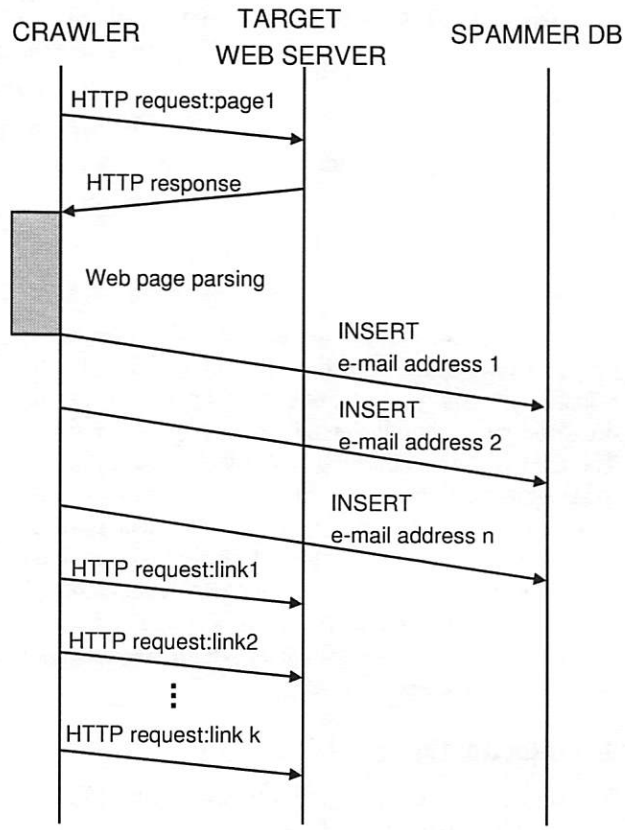



Figure 1: E-mail harvesting

**Operating anonymously.** Spamming activities are illegal in many (but not every) countries, thus *anonymity* is one of the most important goals pursued by a spammer. Figure 2 shows the actions undertaken by a spammer to gain anonymity when sending e-mails.

A common technique to hide traces of malicious activities is the use of an *open relay*, that is an SMTP server which does not need authentication to forward e-mail messages. The open relay is contacted by the spammer, receives the e-mail message, and forwards it to the destination SMTP server. This path is detailed through crosshatch arrows in Figure 2. In theory, a single open relay is sufficient to provide anonymity because every SMTP request appears to be coming from the open relay IP address. However, if a spammer contacts an open relay that is configured to log any activity, the IP address of the spamming machine gets stored in the open relay log files. Using a single open relay is considered a "risky" operation for the spammer.

For this reason, spammers often use one or more *open proxies* (which form a *proxy chain*) to hide their activ-

ities. An open proxy is an intermediary node that forwards traffic between a client and a server without the need of authentication. As shown by straight-arrow path in Figure 2, the spammer establishes a TCP connection with the first open proxy (*open proxy 1*). Next, the spammer uses this connection to establish a new proxy connection with another open proxy (e.g., *open proxy 2*) through the *CONNECT* method. In this way, a chain of proxy connections is built, until *open proxy n*. Finally, the spammer uses the proxy chain to send an e-mail message to an open relay, which is forwarded to the destination SMTP server.

The use of open proxy chains makes it much more difficult to trace spammers back, because, even if the logs of every open proxy are available, the whole path of requests has to be reconstructed. The problem is augmented because public lists of open proxies can be easily found online, for instance at [23].

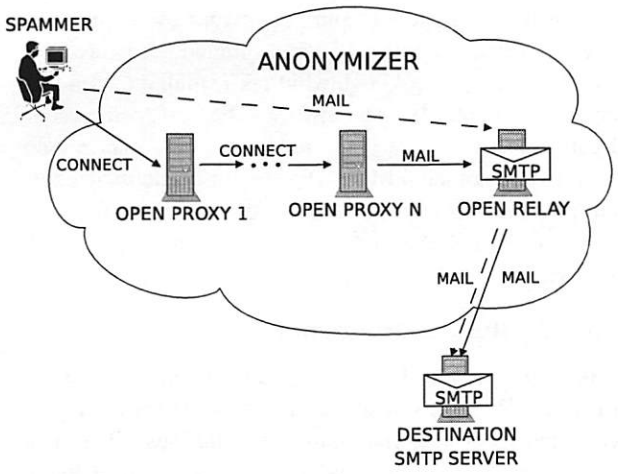



Figure 2: Sending spam anonymously

## 3 Architecture of HoneySpam

Once the principal activities of a spammer are detailed, in this section we outline the requirements that a system fighting against spam sources has to satisfy. Next, we propose the architectural design and some implementation details of HoneySpam.

### 3.1 Requirements of an anti-spam system

To effectively fight against spam sources, it is necessary to contrast the operations described in Section 2. In particular, we identify the following three goals.

**Reduce the efficiency of crawlers.** Spammers rely on up-to-date lists of valid e-mail addresses in order to reach as many recipients as possible. Thus, it is crucial to reduce the efficiency of the e-mail harvesting process. This goal can be obtained in several ways, as shown below.

A first step to reduce the effectiveness of crawlers consists in the possibility of forcing them into a (possibly endless) loop of Web page retrievals. In particular, if each Web page in the loop does not contain any parsable e-mail address, the final goal of the crawler (that is, collecting valid e-mail addresses) is defeated.

A further damage that can be inflicted to crawlers is the possibility of populating the Web pages with invalid addresses (*poisoning*). These are parsed by the crawlers and are usually fed to the spammer databases. The end result is a database that contains many invalid entries (*polluted database*). The drawback of possibly increased network traffic due to the spammer attempts to contact all those fake addresses can be used to better trace spam sources.

Web pages are browsed not only by malicious crawlers, but also by normal ones (e.g., *Googlebot*). To permit legitimate crawling, the *robot exclusion protocol* [5] has to be implemented at the Web server. It tells crawlers which portions of the site are available to indexing purposes and which are not. The vast majority of legitimate crawlers complies to this protocol and is not involved in the Web page retrieval loop.

**Identify spammers.** To identify spammers, it is necessary to encourage them to use honeypot services to their advantages. This is done through the deployment of fake servers, such as open proxies and open relays.

Letting spammers use honeypot services is not enough. To ensure traceability of their actions, *logging* must be enabled for every deployed service (open proxy, open relay, Web server).

A further measure that helps in the identification is the use of valid e-mail addresses in the Web pages returned by the fake Web server. These addresses are handled by a honeypot SMTP server, which logs every activity and every message. This information is parsed to obtain valuable information about the spam sources.

**Block spam e-mails** To reduce the end effects of spam that is, the deliver of unsolicited messages, the associated traffic has to be blocked. A good anti-spam tool must not block valid e-mail messages (*false positives*) and should pass the least amount of unsolicited messages (*false negatives*).

### 3.2 Architecture

Figure 3 shows the architecture of HoneySpam. The framework is placed into the DMZ area of an enterprise network. HoneySpam includes several emulated components, each one dedicated to fight one specific aspect of spamming. We distinguish the following components: *fake Web servers* (to allow e-mail harvesting), *fake open proxies*, *fake open relays* (to provide spammers with services that can be used to their advantage), *destination*

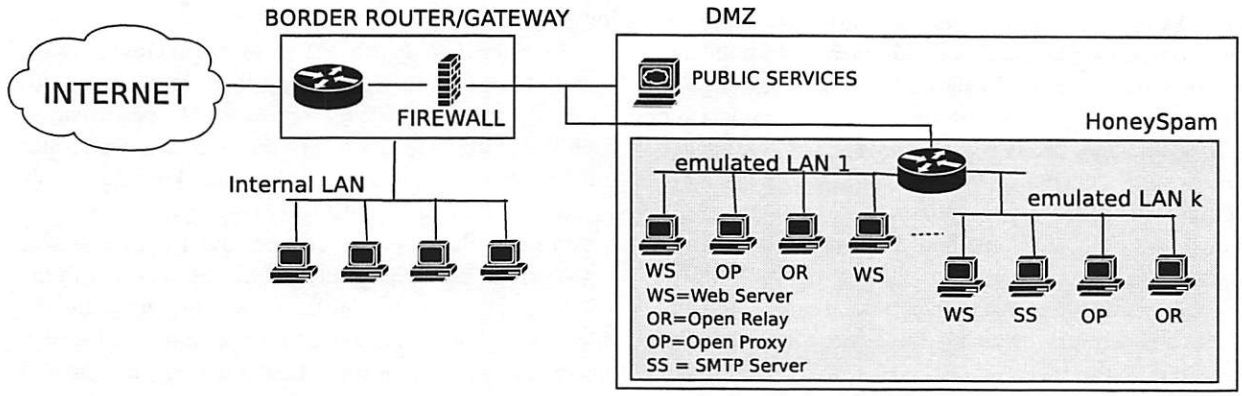



Figure 3: Architecture of HoneySpam

*SMTP servers* (to increase spammers traceability). Service emulation is obtained through honeypots. In the following we detail the design choices and some implementation details regarding honeypots and each considered component.

## 3.3 Honeypots

The first design choice concerns the deployment of honeypots. Many alternatives exist.

A possibility is to use real software on real hardware, for example an instance of an Apache Web server running on a node. This solution, though feasible, is definitely not advisable, since software typically has security bugs, which could be used to penetrate the host in order to obtain valuable information, to install "unwanted" services, even to obtain a privilege escalation. Besides making a service unavailable, a compromised host could also be used to attack third-party nodes. In this case, it is often necessary to install both the operating system and the applications from scratch.

Another possibility is to use real software on emulated hardware, for instance by using User-Mode Linux (UML), as explained in [24]. UML is an open source virtual machine. It allows to run multiple instances of Linux, as a user process, on the same system at the same time. Real services are executed on emulated hosts. Using emulated hardware instead of real hardware is a better approach, because if the host is compromised, it is only necessary to replace the copy of the emulated host and to restart it to obtain the previous situation. Still there is the possibility to use the security holes of the UML software to acquire superuser privileges and to drive attacks to third party hosts.

Finally, it is possible to run emulated software on both real and emulated hardware. Depending on the amount of the available system nodes, emulated services are executed on real or emulated hardware. In this way no security holes can be exploited, simply because there is no real service running, only an emulation.

This possibility seems the most promising for implementing honeypots, because it is the most immune solution from exploits, which must be avoided at all costs. The previous considerations motivate the implementation of the honeypots of our framework by using emulated software on both real and emulated hardware. We use the `honeyd` tool [14], which is a small daemon that creates virtual hosts on a network. The hosts can be configured to run arbitrary services (e.g., an Apache Web server, a sendmail SMTP server), on predefined operating systems (Windows 2000 Pro, GNU/Linux with a 2.4.x kernel). These services are implemented through Perl scripts.

## 3.4 System components

**Fake Web server.** To protect against harvesting we deploy fake Web sites, whose goal is to slow down the process and to pollute the spammer databases. The idea is to provide a (potentially endless) sequence of links, thus catching the crawlers in a very long process. The pages of HoneySpam Web servers provide "special" e-mail addresses that, if used by the spammer, help trace him back. Furthermore, each action is logged in order to understand who is accessing the Web servers.

The *fake Web servers* generate dynamic pages containing many links and e-mail addresses which cannot be identified as fake by common spammer tools [15]. This idea resembles the behavior of Wpoison [6], with some improvements. The number of links, as well as that of the addresses is randomly chosen in a given range (for instance there is a minimum of 2 and a maximum of 20 links per document). We also provide many text files to be randomly used as the body of the html document. Random generation of Web pages greatly reduces determinism and, consequently, the risk of being identified as a honeypot. The e-mail addresses are not randomly chosen, because otherwise one cannot be sure to create a valid address. Instead, the user names are created as a composition of as much information as pos-

sible about the crawler from the Web server it is visiting. We collect the source IP address as well as the timestamp of the request and create a user name of the form `crawler-IP_timestamp`. The domain is randomly chosen among the list of domains belonging to our system. This approach is useful in two ways. First of all these addresses cannot be recognized as fake by software like e-mail Verifier [15], whose goal is to clean polluted e-mail databases. Next, spammers send e-mails to many of our honeypots, thus increasing the effectiveness of traceability.

An interesting aspect concerns the creation of fake links. These are randomly distributed among different Web servers. At each Web server the same script is executed, regardless of the requested resource, except for the requests directed to `robots.txt` to implement the robot exclusion protocol.

**Fake open proxies and relays.** In order to destroy the anonymity of the spammer, we provide virtual open proxies and open relays, whose main goal is to intercept illegal traffic operated by spammers. Connections to our open proxy and open relay servers are logged, to understand where the unwanted traffic is coming from. Furthermore, the traffic is blocked, thus reducing the number of unwanted e-mails reaching target users.

A connection that reaches an open proxy is usually generated by a spammer or another open proxy, in both cases enemy entities. Consequently, the corresponding source IP address is logged (and, optionally, inserted into a black list). We also log the IP address of the following node, since usually it is another open proxy or an open relay [12]. This allows HoneySpamto catch two enemy entities at a time. A connection that reaches an open relay is almost always generated by an open proxy, mainly for anonymity reasons. In this case, the corresponding source IP address is logged (and, possibly, blacklisted).

The *fake open proxies* emulate a subset of the HTTP protocol. Requests made with methods other than GET and CONNECT are answered with an error message. GET requests are answered with a randomly generated page. CONNECT requests to port 25 are internally redirected to an emulated open relay. The motivation behind this redirection is that the spammer may think nothing went wrong and he is connected to the SMTP server he requested, while he actually is connected to one of our honeypots. CONNECT requests to ports other than 25 are served with a "Request Timeout" message.

The *fake open relays* emulate a `sendmail` SMTP server. All the main commands of the SMTP protocol have been implemented, so that no one could notice the difference with a real server. When an e-mail is sent through the open relay, it actually does not reach destination, since all messaged are logged but not forwarded, except the very first one. This is done in order to fool a spammer who sends a first probe message to himself to see if the service is properly running.

**Fake destination SMTP server.** The destination SMTP server is specific to the domains protected by HoneySpam. If the spammer has harvested e-mail addresses from Web servers present in HoneySpam, these addresses will correspond to mailboxes in HoneySpam's destination SMTP servers. The idea is to redirect all the traffic coming to any of the special address to a single mailbox, in which all the messages are logged and stored for analysis and backtracking purposes. The *fake destination SMTP server* is implemented similarly to the open relay; as a matter of fact, the set of SMTP commands is the same. This time the server is configured not to send any message, if requested to.

## 4 Possible countermeasures to HoneySpam

In this section we present a brief survey of malicious activities that can be directed to HoneySpam and related responses.

**Honeypot identification.** One on the main problems when using honeypots is the possibility to be discovered. Attackers use *footprinting* techniques [4, 9] in order to understand whether the services they are using are real or emulated. The `honeyd` framework is extremely useful to this purpose, since it is capable of returning different messages based on which operating system and services it is emulating. We tested its capabilities of emulating network topologies with many tools, including `ping`, `hping`, `traceroute`. All IP addresses emulated by the system are reachable through these tools, thus we can say `honeyd` is effectively able to emulate network topologies. Furthermore, we tested the capability of emulating in a proper way the features of various operating systems (Windows 2000 Professional, and Linux with a 2.4.7 kernel, among others) and of the services (a sendmail SMTP server, an Apache Web server). Network scanners such as `nmap` and `amap` have been fooled by `honeyd`. Indeed, they were not able to recognize that the operating systems and the services were emulated and not real. Thus, we can conclude that, with the tools usually adopted by spammers, it is rather difficult to recognize our servers as honeypots.

Besides white hats, spammers and attackers also maintain their blacklists (in this case, of honeypots); they are called *reverse blacklists*. In the actual implementation, HoneySpam is not able to address this problem, because each emulated server uses only one IP address. Each blacklisted server is no longer available for honeypotting. A possible improvement, which does not solve this problem entirely, is to use as many IP addresses as possible. An authoritative DNS server will

map entire IP ranges into one hostname in a random way. Thus, if one of the IP addresses of a host is blacklisted, the host will still be reachable thanks to the other IP addresses it is related to.

**Intrusion.** Another possible malicious action is to penetrate the host in order to get valuable information, to install "unwanted" services or even to attack a third party. This is usually accomplished by using a software or protocol vulnerability. For instance, if the attacker gets to understand that the servers are running `honeyd`, he could take advantage of its security holes to penetrate one of HoneySpam's hosts. Removing all security vulnerabilities in `honeyd` is difficult, but these risks may be kept limited through simple actions. For example, `honeyd` has to be run with superuser privileges in order to let it manage networking; this is quite dangerous. If, on the other hand, `honeyd` runs in a restricted environment (such as a *chroot* jail), the privileges obtained through an exploit are limited only to the restricted area. As a consequence, the attacker is prevented from having access to sensible information such as system configuration files. To improve protection against intrusion attacks, HoneySpam can be integrated with a file alteration monitoring service, such as [28]. Its main purpose is to check for the creation of new files, which is one of the most common steps in installing new unwanted software. Changing or just reading existing files should also be cared, with the exception of the log files, because they are constantly changed by HoneySpam.

## 5 Related Work

The idea of using honeypots to fight spam is not quite novel. There are many works in literature describing how to use honeypots to detect suspicious traffic in order to understand how black hats behave, which include (but are not limited to) spam traffic detection.

A recent work describing the main steps followed by spammers in sending e-mails is presented in [12]. The author also explains how it is possible to use honeypots to fight the various spammer activities, for instance by deploying fake Web servers in order to pollute the spammers databases, after the e-mail harvesting process, or by deploying fake open proxies and relays in order to trace the spammer back. The author explains how to use real Web servers and how to turn them into honeypots.

In [14] Provos suggests to implement fake open proxies and relays through the `honeyd` framework, in an emulated network environment. HoneySpam not only implements all the suggestions of this paper, but also adds an anti-harvesting mechanism that poisons the spammer databases. The goal is to drive spammer activity to other honeypots of the HoneySpam framework.

In [10] a mobile honeypot mechanism is proposed, that allows unwanted traffic to be detected significantly close to the origin of spam. This is obtained through a strict cooperation among Autonomous Systems (ASs). The main goal of Mohonk is to detect and possibly block the routing of messages through the use of dark address spaces. HoneySpam tries to reach the same goal, that is to block spam close to its source.

Real software can also be used to provide honeypots. As an example, a description on how to use *sendmail* as a honeypot SMTP server is presented in [20]. In [1], an open proxy honeypot (Proxypot) is deployed through an Apache Web server compiled with additional security modules. Wpoison [6] is a tool that attempts to solve the e-mail harvesting problem. It is a CGI script that generates randomized invalid addresses and pseudo-hyperlinks. The goal is to pollute the spammers databases and to capture the crawlers in a possibly endless loop. There are two main differences with the approach of HoneySpam: the e-mail addresses are bogus, thus preventing the interaction with further honeypot facilities, such as HoneySpam's destination SMTP servers. Furthermore, being it a CGI script, this tool has to be run on a real Web server, which is a security treat.

## 6 Future work

Though it is far from being perfect, HoneySpam can be extended to improve its effectiveness in fighting spam sources. There are two possible directions to look for improvement.

**Scalability and fault-tolerance** Given the increasing amount of attacks, a single honeypot is more likely to become the bottleneck of the entire framework. Honeypot overloads have to be definitively avoided, since an overloaded host loses appreciation to spammers, and its attracting potential decreases. A future direction towards scalability consists in the replication of the honeypots, both at the local and at the geographical level.

**Limiting the network throughput of spammers** Another way to avoid overloads is to take advantage of traffic shapers, which limit both incoming and outgoing bandwidth to given thresholds. Traffic shaping also carries another benefit: it helps slowing down the throughput associated to spammer activities. In order to better emulate the network conditions, variable routing delays should be considered. In this way it is much more difficult for a spammer to recognize that the traffic is artificially slowed down, since these network conditions resemble a real environment.

## 7 Conclusions

In this paper we have shown the design and implementation of a fully working anti spam framework. Through the deployment of honeypot services, we are able to face the main spammers' malicious actions. The e-mail harvesting phase is slowed down and the addresses

databases are polluted with special e-mail addresses, which lead the spammer to interact with other honeypot services (destination SMTP servers). This improves spammer traceability. We also deploy honeypot open proxies and relays, with the goal of intercepting and blocking unwanted traffic. Furthermore all interactions are logged with analysis purposes to increase the spammer traceability. Finally, we take into account the main attacks spammers could drive against our framework, and how we can face them. We are currently testing the implementation of the HoneySpam framework. We are running `honeyd` on a few machines to test the functionality and effectiveness of the implemented services.

## Acknowledgements

The authors acknowledge the financial support of the FIRB project "Wide-scalE, Broadband, MIddleware for Network Distributed Services" (Web-MiNDS).

## References

[1] BARNETT, R. C. Open Proxy Honeypots, Mar 2004. http://honeypots.sourceforge.net/open_proxy_honeypots.pdf.

[2] CENTER FOR DEMOCRACY & TECHNOLOGY. Why Am I Getting All This Spam?, Mar 2003. http://www.cdt.org/speech/spam/030319spamreport.pdf.

[3] CHANNEL MINDS. Spam Traffic Risen 40% Since November Says Email Systems, Feb 2005. http://www.channelminds.com/article.php3?id_article=2464.

[4] COREY, J. Advanced Honey Pot Identification And Exploitation, Jan 2004. http://www.phrack.org/fakes/p63/p63-0x09.txt.

[5] COSTER, M. A Method for Web Robots Control, Nov 1996. http://www.robotstxt.org/wc/norobots-rfc.html.

[6] E-SCRUB TECHNOLOGIES, INC. Wpoison, 2000. http://www.monkeys.com/wpoison/.

[7] ERIDANI STAR SYSTEM. MailStripper, 2005. http://www.eridani.co.uk/MailStripper/.

[8] HANNA, J. Anti-Spam SMTP Proxy, 2004. http://assp.sourceforge.net/.

[9] KOHNO, T., BROIDO, A., AND CLAFFY, K. Remote physical device fingerprinting. In *Proc. of the 2005 IEEE Symposium on Security and Privacy* (May 2005).

[10] KRISHNAMURTHY, B. Mohonk: Mobile honeypots to trace unwanted traffic early. In *NetT '04: Proceedings of the ACM SIGCOMM workshop on Network troubleshooting* (Aug-Sep 2004).

[11] OPEN FIELD SOFTWARE. Ella for Spam Control, 2005. http://www.openfieldsoftware.com/.

[12] OUDOT, L. Fighting Spammers With Honeypots: Part 1 and 2, Nov 2003. http://www.securityfocus.com/infocus/1747.

[13] PRINCE, M., KELLER, A. M., AND DAHL, B. No-Email-Collection Flag. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)* (Jul 2004).

[14] PROVOS, N. A Virtual Honeypot Framework. In *Proc. of 13th USENIX Security Symposium* (Aug 2004).

[15] SEND-SAFE. Email Verifier, 2005. http://www.send-safe.com/verifier.php.

[16] SERVICE STRATEGIES INC. MailSWAT SPAM Filter, 2003. http://www.ssimail.com/SMTP_spam_filter.htm.

[17] SNYDER, J. What is a false positive?, Dec 2004. http://www.networkworld.com/reviews/2004/122004spamside3.html.

[18] SOPHOS. MailMonitor, 2005. http://www.sophos.com/products/es/gateway/.

[19] SOPHOS. PureMessage, 2005. http://www.sophos.com/products/es/gateway/.

[20] SPENCER, B. Fighting Relay Spam the Honeypot Way, Mar 2002. http://www.tracking-hackers.com/solutions/sendmail.html.

[21] SUBMIT SERVICES AND WEB WORLD DIRECTORY. MailWasher PRO, 2005. http://www.submitservices.com/mailwasher.html.

[22] THE APACHE SOFTWARE FOUNDATION. The Apache SpamAssassin Project, 2005. http://spamassassin.apache.org/.

[23] THE ATOMINTERSOFT TEAM. AliveProxy, 2005. http://www.aliveproxy.com/.

[24] THE HONEYNET PROJECT & RESEARCH ALLIANCE. Know your Enemy: Learning with User-Mode Linux, Dec 2002. http://www.honeynet.org/papers/uml/.

[25] THE SPAMHAUS PROJECT. SpamHaus Block List. http://www.spamhaus.org/sbl/index.lasso.

[26] UNIVERSITY OF ALBERTA. SpamAssassin FAQ. http://www.ualberta.ca/HELP/email/spamfaq.html.

[27] UNSPAM, LLC. Project Honey Pot, 2005. http://www.projecthoneypot.org.

[28] WARDLE, M. FAM - File Alteration Monitor, 2004. http://savannah.nongnu.org/projects/fam/.

# Improving Spam Detection Based on Structural Similarity

Luiz H. Gomes*, Fernando D. O. Castro,
Virgílio A. F.Almeida, Jussara M. Almeida, Rodrigo B. Almeida
*Computer Science Dept., Universidade Federal de Minas Gerais, Belo Horizonte - Brazil*
*{lhg,fernando, virgilio,jussara, barra}@dcc.ufmg.br*

Luis M. A. Bettencourt
*Computer and Computational Sciences, Los Alamos National Laboratory, Los Alamos - USA*
*lmbett@lanl.gov*

## Abstract

We propose a new spam detection algorithm that uses structural relationships between senders and recipients of email as the basis for spam detection. A unifying representation of users and receivers in the vectorial space of their contacts is constructed, that leads to a natural definition of similarity between them. This similarity is then used to group email senders and recipients into clusters. Historical information about the messages sent and received by the clusters is obtained by forwarding messages to an auxiliary spam detection algorithm and this information is used to reclassify messages. In the framework proposed, our algorithm aims at correcting misclassifications from an auxiliary algorithm. A simulation is performed based on actual data collected from an SMTP server from a large University. We show that our approach is able reduce false positives, produced by the auxiliary classification algorithm, up to about 60%.

## 1 Introduction

The relentless rise in spam email traffic, now accounting for about 83% of all incoming messages, up from 24% in January 2003 [14], is becoming one of the greatest threats to the use of email as a form of communication. Spam is also increasingly at the root of major security breaches as more viruses, worms and other malicious software makes use of spam messages to spread throughout the Internet.

A major problem in detecting spam stems from active adversarial efforts to thwart classification. Spam senders use a multitude of techniques based on knowledge of current algorithms, to evade detection. These techniques range from changes in the way text is written to frequent changes in elements, such as user names, domains, subjects, etc. Although such evasion strategies are usually naturally understood by humans, it is hard to automatically analyze them.

A central question that remains unanswered is: what are the identifiers of spam that are most costly to change, from the point of view of the spammer? The limitations of attempts to recognize spam by analyzing content are emphasized in [7]. Content-based techniques [15, 22, 16] have to cope with the constant changes in the way spammers generate their solicitations. The structure of the target space for these solicitations tends, however, to be much more stable since spam senders still need to reach recipients in order to be effective. Specifically, by structure we mean the space of recipients targeted by a spam sender, as well as the space of senders that targets a given recipient, i.e. the contact lists of a user. The contact lists, or subsets thereof, can then be thought of as a (dynamical) signature of spam senders and recipients. Additionally by constructing a similarity measure in these spaces we can track how lists evolve over time, by addition or removal of addresses.

In this paper, we propose and evaluate an algorithm for improving spam detection that uses structural relationships between senders and recipients as the basis for the detection of spam messages. The algorithm works in conjunction with another spam classifier (hereafter called auxiliary algorithm), necessary to produce spam or legitimate mail tags on past senders and receivers, which in turn are used to infer new ones, through structural similarity. The key idea is that the set of distinct recipients that spammers and legitimate users send messages to, as well as the set of distinct senders from which users receive messages from (which, in both cases, we call *contact lists*), can be used as identifiers of senders and recipients in email traffic [19, 11, 12]. We show that the application of our structural algorithms over the auxiliary classifier's results leads to the correction of a number of misclassifications.

This paper is organized as follows: Section 2 presents the methodology used to handle email data. Our struc-

*Luiz H. Gomes is supported by Banco Central do Brasil.

tural algorithm is described in Section 3. We present the characteristics of the workload studied in section 4, as well as the classification results obtained with our algorithm over this data set. Related work is presented in Section 5 and conclusions and future work in Section 6.

## 2 Modeling Similarity Among Email Senders and Recipients

Our proposed spam detection algorithm exploits the structural similarities in groups of senders and recipients of email. This section introduces a unifying modeling framework of individual email users and a metric that captures the similarity between them.

Our basic assumption is that, in legitimate and spam traffics, users have a list of peers they often have contact with (i.e., they send/receive an email to/from), as can be seen in Section 4. In legitimate traffic, contact lists stem from social relationships. On the other hand, the lists created by spammers to distribute their solicitations are guided by business opportunities and, generally, do not reflect any form of social interaction. Contact lists certainly change over time. However, we expect them to be much less variable than other identifiers commonly used for spam detection, such as the presence of certain keywords in the email content or its size and encoding. In other words, we expect contact lists to be an effective basis for detecting spam.

We start by representing an email user as a vector in a multi-dimensional vectorial space created out of all possible contacts. We represent email senders and recipients separately. We then use vectorial operations (the normalized inner product in the sender/recipient spaces) to express the similarity among multiple senders/recipients. Finally, similarity is used to associate users into clusters. Note that the term email user is used throughout this work to denote any identification of an email sender/recipient (e.g., email address, domain name, SMTP relay, etc).

Let $N_r$ be the number of distinct recipients. We represent a sender $s_i$ as a $N_r$ dimensional vector, $\vec{s_i}$, defined in the vectorial space of email recipients. The $n$-th dimension (representing recipient $r_n$) of $\vec{s_i}$ is defined as:

$$\vec{s_i}[n] = \begin{cases} 1, & \text{if } s_i \rightarrow r_n \\ 0, & \text{otherwise} \end{cases}, \quad (1)$$

where $s_i \rightarrow r_n$ indicates that sender $s_i$ has sent at least one email to $r_n$ recipient.

Similarly, we define $\vec{r_i}$ as a $N_s$ dimensional vector representing recipient $r_i$, where $N_s$ is the number of distinct senders being considered. The $n$-th dimension of this vector is analogously set to 1 if recipient $r_i$ has received at least one email from $s_n$.

We next define the similarity between two senders $s_i$ and $s_j$ as the cosine of the angle between their vectorial representation ($\vec{s_i}$ and $\vec{s_j}$). The cosine is a well known metric that has been successfully employed in several application areas, including document similarity in information retrieval systems [3, 20] and intrusion detection [21]. This similarity metric is computed as:

$$sim(s_i, s_j) = \frac{\vec{s_i} \circ \vec{s_j}}{|\vec{s_i}||\vec{s_j}|} = cos(\vec{s_i}, \vec{s_j}), \quad (2)$$

where $\vec{s_i} \circ \vec{s_j}$ represents the internal product of the vectors and $|\vec{s_i}|$ is the norm of $\vec{s_i}$. This metric varies between 0, when senders do not share any recipient in their contact lists, and 1, when senders have identical contact lists and thus have the same representation. The similarity between two recipients is similarly defined.

We note that our similarity metric has different interpretations in legitimate and spam traffics. In legitimate email traffic, it represents social relationships that consist of interactions with the same group of users, whereas in the spam traffic, a great similarity probably represents the use of different identifiers by the same spammer or the sharing of distribution lists by distinct spammers.

Finally, we can use our vectorial modeling approach to represent a cluster of senders or recipients. A sender cluster $sc_i$, represented by vector $\vec{sc_i}$, is computed as the vector sum of its elements, that is:

$$\vec{sc_i} = \sum_{s_j \in sc_i} \vec{s_j} \quad (3)$$

The similarity between a sender $s$ and an existing cluster $sc_i$ can be assessed by extending Equation 2 as follows:

$$sim(sc_i, s) = \begin{cases} cos(\vec{sc_i} - \vec{s}, \vec{s}), & \text{if } s \in sc_i \\ cos(\vec{sc_i}, \vec{s}), & \text{otherwise} \end{cases} \quad (4)$$

We note that the vectorial representation of a sender $\vec{s}$ and of the the sender's cluster may change over time as new emails are considered. In order to accurately estimate the similarity between a sender $\vec{s}$ and a sender cluster $\vec{sc_i}$ to which $\vec{s}$ currently belongs to, we first remove $\vec{s}$ from $\vec{sc_i}$, and then take the cosine between the two vectors ($\vec{sc_i} - \vec{s}$ and $\vec{s}$). This guarantees that the previous classification of a user does not influence its new classification. Recipient clusters and similarities are defined analogously.

## 3 Structural Similarity Algorithm

This section introduces our new email classification approach, which exploits the similarity between email senders and recipients for their association into clusters, which are tagged by historical information. Our algorithms is designed to work together with any existing

spam detection or filtering technique. Our goal is to provide a significant reduction of false positives (i.e., legitimate emails wrongly classified as spam), which can be as high as 15% in current filters [2].

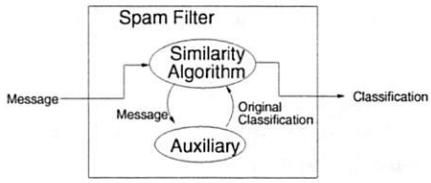



Figure 1: Architecture Proposed.

The architecture proposed in this paper is shown in Figure 1. A message arrives at the spam detection system and is directed to the structural similarity algorithm. This algorithm first sends the message to the auxiliary detection algorithm to retrieve a first classification of that message. Based on this classification, on the cluster formed by senders and recipients, and on previous historical information, our algorithm generates a new classification, which may or may not coincide with the original classification provided by the auxiliary. The idea is to use the classification provided by the auxiliary method to build an incremental historical knowledge base that becomes more representative as more messages are processed.

**for all** arriving message $m$ **do**
  $mClass$ =classification of $m$ by auxiliary detection method;
  $sc$ =find cluster for $m.sender$;
  Update spam probability for $sc$ using $mClass$;
  $P_s(m)$ =spam probability for $sc$;
  $P_r(m) = 0$;
  **for all** recipient $r \in m.recipients$ **do**
    $rc$ =find cluster for $r$;
    Update spam probability for $rc$ using $mClass$;
    $P_r(m) = P_r(m)$+spam probability for $rc$;
  **end for**
  $P_r(m) = P_r(m)/size(m.recipients)$
  $SP(m)$ = compute spam rank based on $P_s(m)$ and $P_r(m)$;
  **if** $SP(m) > \omega$ **then**
    classify $m$ as spam;
  **else if** $SP(m) < 1 - \omega$ **then**
    classify $m$ as legitimate;
  **else**
    classify $m$ as $mClass$;
  **end if**
**end for**

Algorithm 1: New Algorithm for Email Classification

A description of the cluster-based algorithm is shown in Algorithm 1. This algorithm maintains sets of sender and recipient clusters, assembled from structural similarity, as defined in Equation (4). A sender/recipient of an incoming email is added to the sender/recipient cluster that is most similar to it, provided that their similarity exceeds a given threshold $\tau$. Thus, $\tau$ defines the minimum similarity a sender/recipient must have with a cluster to be assigned to it. Varying $\tau$ allows us to create more or less tightly knit clusters. If no similar cluster can be found, a new single-user cluster is created.

The sets of sender and recipient clusters are updated at each new email arrival. Recall that to determine the cluster of a previous classified user we first remove the user from its current cluster and then assess its similarity to each existing cluster. Thus, single-user clusters tend to be reduced as more emails are processed, except possibly for users that appear only very sporadically.

A probability of sending/receiving spam messages is assigned to each sender/recipient cluster. We refer to this measure as the *cluster spam probability*. We calculate the spam probability of a sender/recipient cluster as the average spam probability of its elements, which, in turn, is estimated from the frequency of spams sent/received by each of them in the past. Therefore, our scheme uses the result of the email classification performed by the auxiliary algorithm on each arriving email $m$ ($mClass$ in Algorithm 1) to continuously update cluster spam probabilities.

Let us define the probability of a message $m$ being sent by a spammer, $P_s(m)$, as the probability of its sender's cluster send a spam. Similarly, let the probability of an email $m$ being addressed to users that receive spam, $P_r(m)$, as the average spam probability of all of its recipients' clusters. Our algorithm uses $P_s(m)$ and $P_r(m)$ to compute a number that expresses the chance of email $m$ being spam. We call this number the *spam rank* of email $m$, denoted by $SR(m)$. The idea is that emails with large values of $P_s(m)$ and $P_r(m)$ should have large spam ranks and thus should be classified as spam messages. Similarly, emails with small values of $P_s(m)$ and $P_r(m)$ should receive low spam rank and be classified as legitimate email.

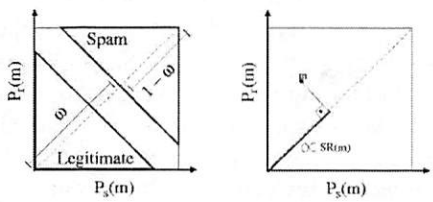



Figure 2: Spam Rank Computation and Email Classification for the Cluster-based Algorithm.

Figure 2 shows a graphical representation of the computation of the spam rank for a message. We first normalize the probabilities $P_s(m)$ and $P_r(m)$ by a factor of $\sqrt{2}$, so that the diagonal of the square region defined in the bi-dimensional space is equal to 1 (see Figure 2-left). Each email $m$ is represented as a point in this square. The spam rank of $m$, $SR(m)$, is then defined as the length of the segment starting at the origin (0,0) and ending at the

projection of $m$ on the diagonal of the square (see Figure 2-right). With these definitions the spam rank varies between 0 and 1.

The spam rank $SR(m)$ is then used to classify $m$ as follows: if it is greater than a given threshold $\omega$, e.g. $\omega \geq 0.5$, the email is classified as spam; if it is smaller than $1 - \omega$, it is classified as legitimate email. Otherwise, we can not precisely classify the message, and we rely on the initial classification provided by the auxiliary algorithm. The parameter $\omega$ can be tuned to determine the precision of our classification. Graphically, emails are classified according to the marked regions shown in Figure 2-left. The two identical triangles represent the regions where our algorithm is able to classify emails as either spam (upper right) or legitimate email (lower left).

## 4 Experimental Results

In this section we describe our experimental results. We first present the most relevant details of our workload, followed by the quantitative results of our approach.

### 4.1 Workload

Our email workload consists of anonymized and sanitized SMTP logs of incoming emails to a large university in Brazil, with around 22,000 students. The server handles all emails coming from domains outside the university, sent to students, faculty and staff with email addresses under the university's domain name [1].

The central email server runs Exim email software [10], the AMaViS virus scanner [1] and the Trendmicro Vscan anti-virus tool [17]. A set of pre-acceptance spam filters (e.g. black lists, DNS reversal) blocks about 50% of the total traffic received by the server.

The messages not rejected by the pre-acceptance tests are directed to Spam-Assassin [16]. Spam-Assassin is a popular spam filtering software that detects spam messages based on a changing set of user-defined rules. These rules assign scores to each email received, based on the presence in the subject or in the email body of one or more pre-categorized keywords. Spam-Assassin also uses other rules based on message size and encoding. Highly ranked messages according to these criteria are flagged as spam.

We analyze a log collected from 01/02/2004 to 01/10/2004. Our logs store the header of each email (i.e. containing sender, recipients, size , date, etc.) that passes the pre-acceptance filters, along with the results of the tests performed by Spam-Assassin and the virus scanners. We also have the full body of the messages that were classified as spam by Spam-Assassin. Table 1 summarizes our workload.

| Measure | Non-Spam | Spam | Aggregate |
|---|---|---|---|
| # of emails | 168,352 | 153,494 | 321,846 |
| Size of emails | 7.5 GB | 0.8 GB | 8.3 GB |
| # of distinct senders | 12,738 | 15,325 | 22,809 |
| # of distinct recipients | 23,849 | 25,383 | 34,065 |

Table 1: Summary of the Workload

By visually inspecting the list of sender *user names* [2] in the spam component of our workload, we found that a large number of them corresponded to a random sequence of characters, suggesting that spammers tend to change user names as an evasion technique. Therefore, for the experiments presented below we identified the sender of a message by his/her domain [3] while recipients were identified by their full address.

### 4.2 Classification Results

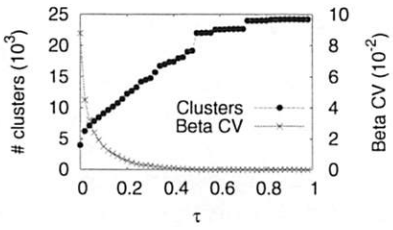



Figure 3: Number of Email User Clusters and Beta CV vs. $\tau$.

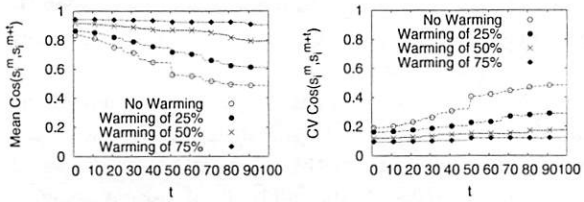
The results shown in this section were obtained through the simulation of the algorithm proposed in this paper over the set of messages in our logs. The implementation of the simulator makes use of an inverted list [20] approach for storing information about senders, recipients and clusters that is effective both in terms of memory and processing time. The classification rate of the simulations were both higher than the peak rate observed over the workload collection time [12].

The number and quality of the clusters generated through our similarity measure are the direct result of the chosen value for the threshold $\tau$ (see Section 3). In order to determine the best parameter value the simulation was executed several times for varying $\tau$.

Figure 3 shows how the beta CV (Coefficient of Variation) for the clusters and the number of clusters vs. $\tau$. Beta CV denotes the intra CV/inter CV for the clusters. Whereas the intra CV measures the coefficient of variation for the similarities intra-cluster, the inter CV measure the similarities between different clusters. Thus, beta CV is a measure of the quality of the clusters generated. The more stable the beta CV the better quality in terms of the grouping obtained [13]. There is one clear point of stability in the curve at $\tau = 0.5$ (Figure 3(a)).

Moreover, the number of clusters generated also stabilize at $\tau = 0.5$ (Figure 3(b)). This is the value we adopt for the remaining of the paper. Although other values of $\tau$, above 0.5 would also be appropriate, the value of $\tau = 0.5$ results in a large number of non unitary clusters, allowing us to evaluate the benefits of the clustering of senders and recipients.

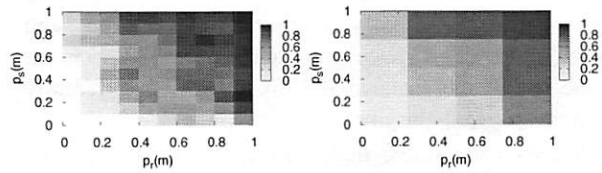
Our approach is motivated by two hypothesis. First, contact lists of email users provide an effective means for identifying them. Second, messages can be more accurately classified as spam or not based on the probabilities of sending/receiving spams of the cluster that their sender/recipients belong to.




(a) Mean Cosine (b) CV of Cosine

Figure 4: Spam Senders Identification Stabilization

In order to show that contact lists provide an effective user identification, we analyze how sender/recipient vectorial representations change over time, as new messages arrive in the system. In this analysis, we consider a warm-up period containing a certain fraction of the messages, during which sender/recipients are updated. We define $\vec{s_i^m}$ as the vectorial representation of sender $i$ at a point in time when it had sent $m\%$ of its own messages ($m$ is larger than the warm-up period). We use the similarity between the representations of sender $i$ in two points in time, spaced by a certain fraction $t$ of messages sent, as a measure of how stable the sender identification is over "period" $t$. We then analyze how this similarity, given by $cos(\vec{s_i^m}, \vec{s_i^{m+t}})$ evolves as the *step* $t$ increases. Figure 4(a) shows average similarity measures for the senders in the spam traffic, varying the step from 1% to 100%, for different warm-up periods. Figure 4(b) shows the corresponding measures of coefficient of variation (CV). Note that, as expected, when there is no warm-up period ("No Warming"), fluctuations in the early messages dominate and stabilization is not reached by the end of our workload. In other words, a larger set of messages would be required for stabilization to be reached. As the warm-up period increases, stabilization is reached with a smaller number of messages. In particular, when we consider only the last 25% of the messages for each sender ("Warming of 75%"), sender representation remains, on average, very stable with CV approaching zero. Similar patterns were observed for legitimate e-mail senders as well as legitimate and spam recipients.

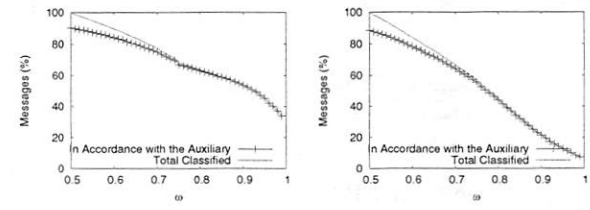



(a) Bin size = 0.10 (b) Bin size = 0.25

Figure 5: Number of Spam Messages by Varying Message Spam Probabilities for Different Bin Sizes.

Next, we investigate whether the second hypothesis hold. We plot, in figure 5, the fraction of spam messages in our workload for different values of $P_s(m)$ and $P_r(m)$ grouped based on a discretization of the full space represented in the plot. This space is subdivided into smaller squares of the same size called bins, the darker the gray scale the greater the number of spams in each bin. Clearly, spam and legitimate messages are located on the top-right and bottom-left regions of the spectrum as we have hypothesized in Section 3. There is, however, an intermediate region in the middle where we cannot satisfactorily determine the classification. This is why it becomes necessary to vary $\omega$. One should adjust $\omega$ based on the level of confidence it has on the auxiliary algorithm.

Figure 5 shows that messages addressed to recipients that have high $P_r(m)$ tend to be spam more frequently than messages with the same value of $P_s(m)$. Analogously, messages with low $P_s(m)$ have higher probability of being legitimate messages.




(a) Legitimate (b) Spam

Figure 6: Messages Classified in Accordance With to the Auxiliary Algorithm and the Total Number of Messages Classified by Varying $\omega$

Because our algorithm makes use of an auxiliary spam

detection algorithm - e.g. SpamAssassin. Therefore, we need to evaluate how frequently we maintain the same classification as such an algorithm. Figure 6 shows the the percentage of messages that received the same classification and the total number of classified messages in our simulation by varying $\omega$, considering only messages classified by the auxiliary as legitimate (Figure 6(a)) and spam (Figure 6(b)). The difference between these curves is the set of messages that were classified differently from the original classification.

Figures 6 also shows that, considering the sets of messages originally classified as legitimate and as spam, our algorithm is capable of classifying a larger number of messages from the former than from the latter. Moreover, we are slightly more conservative in the classification of legitimate emails than spams. We conjecture that both phenomena stem from the fact that the rules used by Spam Assassin tend to favor the detection of legitimate messages.

In another experiment, we simulated a different algorithm that also makes use of historical information provided by an auxiliary spam detector described in [19]. The main differences are that it uses historical information of each sender separately and it does not use recipients information. We built a simulator for this algorithm and executed it against our data set. The results show that it was able to classify $85.11\%$ of the messages in accordance with the auxiliary algorithm, while our approach classify more than $95\%$ with $\omega = 0.85$.

We believe that the differences between the original classification and the classification proposed for high $\omega$ values generally are due to misclassifications by the auxiliary algorithm. In our data set we have access to the full body of the messages that were originally classified as spam. We were able to evaluate a fraction of the total amount of false positives (messages that the auxiliary algorithm classify as spam and our algorithm classify as legitimate) that were generated by the auxiliary algorithm. This is important since the cost of false positives is usually believed higher than the cost of false negatives [7].

| Algorithm | % of Misclassifications |
|---|---|
| Auxiliary | 60.33% |
| Our approach | 39.67% |

Table 2: Possible False Positives Generated by the Approaches Studied.

Each of the possible messages classified by the auxiliary as spam and by our algorithm as legitimate was manually evaluated by three people, in order to determine whether such a message was indeed spam. Table 2 summarizes the results for $\omega = 0.85$, 879 messages ($0.27\%$ of the total messages) were manually analyzed. Our algorithm was correct in more $60\%$ of the cases.

Due to the cost of manually classifying messages we can not afford to classify all of the messages categorized as spam by the auxiliary algorithm. However, we evaluate a randomly chosen fraction of the messages classified as spam by the auxiliary and by our algorithm, which represents the total data with a confidence interval of $99\%$ [18]. With $\omega = 0.85$, we found that $15\%$ of the total messages are in this group and we analyzed manually a sample of $3.50\%$ (1,708 emails) of them. We found that $99.9\%$ of the analyzed messages were correctly classified, showing the high precision of our classification.

Moreover, only $0.11\%$ of the total of messages classified as legitimate by the auxiliary were found to be spam by our approach. Consequently, the total number of messages correctly moved from spam class to legitimate class is $47\%$ greater than the number of messages moved from legitimate class to spam class by our algorithm. Moreover, we emphasize that we can not determine the quality of the classification for the messages classified as legitimate by the auxiliary algorithm since we do not have access to the full body of those messages.

## 5 Related Work

Several previous studies have focused on reducing the impact of spam email traffic. The approaches to reduce spam can be categorized into pre-acceptance and post-acceptance methods, based on whether they detect spam before or after accepting messages. Examples of pre-acceptance methods are server authentication [8, 4] and accountability [6]. Post-acceptance methods are mostly based on information available in the body of the messages and include Bayesian filters [15] and collaborative filtering [22].

Recent papers have focused on spam combat techniques based on characteristics of graph models of email traffic [5, 9]. In [5] a graph is created to represent the email traffic captured in the mailbox of individual users. The clustering coefficient of each of these components is used to classify messages as spam or legitimate. The results show that $53\%$ of the messages were precisely classified using the proposed approach. In [9] the authors propose to detect machines that behave as spam senders by analyzing a border flow graph of sender and recipient machines. In contrast to these studies, we propose to use a structural similarity between email senders and recipients to group them into clusters and use the cluster historical information to improve spam detection. Moreover, unlike [5], our approach runs on the ISP level.

None of the existing spam filtering mechanisms are infallible [19, 7]. Their main handicap are false positives and wrong mail classification. In addition, filters must be continuously updated to capture the multitude of mecha-

nism constantly introduced by spammers to avoid filtering actions. The algorithm presented in this paper aims at improving the effectiveness of spam filtering mechanisms, by reducing false positives and by providing information to tune their collection of rules.

## 6 Conclusions and Future Work

In this paper we proposed a new algorithm to improve spam detection based on the structural similarity between contact lists of email users. The idea is that contact lists, integrated over a suitable amount of time, are much more stable identifiers of email users than user names, domains or message contents, which can all be made to vary quickly and widely. The major drawback of our approach is that our algorithm can only group users based on their structural similarity, but has no way of determining by itself if such vector clusters correspond to spam or legitimate email. Thus it must work in tandem with an original classifier. Given this information we have shown that we can successfully separate spam and legitimate email users and that this structural inference can improve the quality of other spam detection algorithms.

Specifically we have implemented a simulator based on data collected from the main SMTP server for a major university in Brazil that uses SpamAssassin. We have shown that our algorithm can be tuned to produce classifications similar to those of the original classifier algorithm and that, for a certain set of parameters, is was capable of correcting false positives.

As future work, we intend to: (i) explore aging mechanisms to update the vectorial identification of senders and recipients over time, and (ii) study the robustness of our approach against different auxiliary classifiers.

## References


[1] AMaViS - Home Page. http://www.amavis.org.

[2] ATKINS, S. Size and Cost of the Problem. In *Proc. 56th IETF Meeting* (San Francisco, California, USA, March 2003).

[3] BAEZA-YATES, R., AND RIBEIRO-NETO, B. *Modern Information Retrieval.* Addison Wesley Longman Publishing Co. Inc., 1999.

[4] BAKER, H. P. Authentication Approaches. In *Proc. 56th IETF Meeting* (San Francisco, California, USA, March 2003).

[5] BOYKIN, P. O., AND ROYCHOWDHURY, V. Leveraging Social Networks to Fight Spam. *IEEE Computer 38*, 4 (April 2005), 61–68.

[6] BRANDMO, H. P. Solving Spam by Establishing a Platform for Sender Accountability. In *Proc. 56th IETF Meeting* (San Francisco, California, USA, March 2003).

[7] CERF, V. G. Spam, Spim, and Spit. *Commun. ACM 48*, 4 (2005), 39–43.

[8] CRANOR, L. F., AND LAMACCHIA, B. A. Spam! *Communications of the ACM 41-8* (August 1998), 74–83.

[9] DESIKAN, P., AND SRIVASTAVA, J. Analyzing Network Traffic to Detect E-Mail Spamming Machines. In *Proc. ICDM Workshop on Privacy and Security Aspects of Data Mining* (Brighton UK, November 2004), pp. 67–76.

[10] Exim Internet Mailer Home Page. http://www.exim.org.

[11] GOMES, L. H., ALMEIDA, R. B., BETTENCOURT, L. M. A., ALMEIDA, V. A. F., AND ALMEIDA, J. M. Comparative Graph Theoretical Characterization of Networks of Spam and Regular Email. http://www.arxiv.org/abs/cs.CR/0504012, March 2005.

[12] GOMES, L. H., CAZITA, C., ALMEIDA, J., ALMEIDA, V. A. F., AND JR., W. M. Characterizing a Spam Traffic. In *Proc. 4th ACM SIGCOMM Conference on Internet Measurement* (Taormina, Italy, 2004), ACM Press, pp. 356–369.

[13] MENASCÉ, D., AND ALMEIDA, V. *Capacity Planning for Web Services: Metrics, Models and Methods.* Prentice Hall Inc., USA, September 2001.

[14] Message Labs Home Page. http://www.messagelabs.co.uk/.

[15] SAHAMI, M., DUMAIS, S., HECKERMAN, D., AND HORVITZ, E. A Bayesian Approach to Filtering Junk E-Mail. Tech. Rep. WS-98-05, AAAI Workshop on Learning for Text Categorization, Madison, Wisconsin, July 1998.

[16] SpamAssassin Home Page. http://www.spamassassin.org.

[17] Trend Micro Home Page. http://www.trendmicro.com.

[18] TRIVEDI, K. S. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* John Wiley & Sons, New York, NY, 2001.

[19] TWINING, R. D., WILLIANSON, M. M., MOWBRAY, M., AND RAHMOUNI, M. Email Prioritization: Reducing Delays on Legitimate Mail Caused by Junk Mail. In *Proc. Usenix Annual Technical Conference* (Boston, MA, June 2004), pp. 45–58.

[20] WITTEN, I. H., BELL, T. C., AND MOFFAT, A. *Managing Gigabytes: Compressing and Indexing Documents and Images.* John Wiley & Sons, Inc., New York, NY, USA, 1994.

[21] XIE, Y., KIM, H., O'HALLARON, D., REITER, M., , AND ZHANG, H. Seurat: A Pointillist Approach to Anomaly Detection. In *Proc. of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID2004)* (Sophia Antipolis, French Riviera, France, September 2004), Springer Verlag, pp. 238 – 257.

[22] ZHOU, F., ZHUANG, L., ZHAO, B., HUANG, L., JOSEPH, A., AND KUBIATOWICZ, J. Approximate Object Location and Spam Filtering on Peer-to-Peer Systems. In *Proc. Middleware* (Rio de Janeiro, Brazil, January 2003), pp. 1–20.


## Notes

[1] Only the emails addressed to two out of over 100 university subdomains (i.e., departments, research labs, research groups) do not pass through the central server.

[2] The part before @ in email addresses.

[3] The part after @ of an email address

# Lightweight Encryption for Email

Ben Adida
*MIT*
ben@mit.edu

Susan Hohenberger
*MIT*
srhohen@mit.edu

Ronald L. Rivest
*MIT*
rivest@mit.edu

## Abstract

Email encryption techniques have been available for more than a decade, yet none has been widely deployed. The problems of key generation, certification, and distribution have not been pragmatically addressed. We recently proposed a method for implementing a Lightweight Public Key Infrastructure (PKI) for *email authentication* using recent developments in identity-based cryptography and today's existing Internet infrastructure.

While this solution works well for email authentication, *email encryption* exhibits a different threat model that requires special treatment. In this work, we discuss how to achieve email encryption and present a realistic deployment and adoption process, while respecting the current functionality and expectations of email.

## 1 Introduction

Email is a mostly insecure communication medium. Email encryption solutions such as the well-known PGP [14] and S/MIME [10] have existed for more than a decade, yet neither has achieved widespread use. This is due, in large part, to the complexity of key management: a user must generate a keypair, certify and distribute his public key, and obtain a validated public key for all intended recipients.

Even identity-based encryption [12], which proposes to compute public keys directly from users' email addresses (or other identity-related strings), presents key management complications. No realistic, practical architecture has been proposed for making use of identity-based encryption in an Internet-wide setting.

Recently, we proposed and implemented a Lightweight PKI [2, 1] to manage keys for email authentication. We now propose extensions to this architecture for the purpose of email encryption. We call this approach Lightweight Encryption.

### 1.1 Prior Key Management Strategies

Public-key encryption has been around for 25 years. In its basic form, it is well understood: a public key allows for encryption, while an associated private (a.k.a. secret) key performs decryption. The complication lies in associating a public key with a user. How does Bob obtain Alice's public key? How can Bob be certain that the public key he has obtained is indeed Alice's, and not some eavesdropper's?

In classic public-key cryptosystems like RSA [11], El Gamal [7], or Cramer-Shoup [5], each user generates a keypair. The association between a public key and an identity is then certified by the digital signature of some authority. With S/MIME [10], these certification authorities form an organizational hierarchy. With PGP [14], an individual trusts a peer-to-peer certificate chain.

In identity-based public-key cryptosystems, first conjectured in 1984 [12] but only fully implemented in 2000 [4], a master authority generates a master keypair $(MPK, MSK)$ and publishes $MPK$ to the world. A user's public key is then the combination of $MPK$ and the string $id\_string$ representing the user's identity. The user's secret key $SK$ is computable from $id\_string$ only by a master authority in possession of $MSK$ who delivers this key securely to the user. Though identity-based schemes simplify user-key management, there remains a domain-key management problem: the $MPK$-domain association must be safely distributed, and the user secret keys must be securely delivered.

### 1.2 The Lightweight PKI

We recently introduced Lightweight PKI, a mechanism for Internet-wide distribution of identity-based public keys for the purpose of email authentication [2, 1]. Each email domain becomes a master authority for an identity-based scheme *of its choosing* and generates a unique master keypair $(MPK, MSK)$. Each $MPK$ is dis-

tributed via the Domain Name System (DNS), as a TXT record associated with the hostname of the domain's Mail Exchange (MX) record. (Interestingly, Lightweight PKI automatically inherits security from any future improvements to DNS [6].) Email users obtain their secret key $SK$ by email-based identification: the master authority delivers the key directly to the user's inbox. Key revocation is handled by using short-lived keys: the identity string of a public key includes an expiration date [2].

Thus, Alice can sign each of her messages with her secret key. Upon receiving a signed email from `alice@wonderland.com`, Bob can look up $MPK^{\texttt{wonderland.com}}$ via the DNS record for `wonderland.com` . Then, Bob can compute Alice's public key using $MPK^{\texttt{wonderland.com}}$ and the string `alice@wonderland.com`, and verify the signature.

Lightweight Signatures strike a practical compromise. They reasonably assume that Alice's mail server will not actively attack Alice. Then, using only the established infrastructures of DNS and email delivery, they make spoofing outgoing email from Alice as difficult as consistently intercepting Alice's incoming email.

## 1.3 Insufficient Security for Encryption

Consider deploying Lightweight PKI for encryption. In this case, a *passive* adversarial incoming mail server could easily decrypt and read a user's encrypted email. Moreover, even if Alice's mail server is honest, the compromise of the master authority's $MSK$ would reveal all prior encrypted emails for *all* users of that domain.

We must take extra precautions to help honest domains secure their master key and honest users protect their privacy. We conclude two necessary principles for encryption:

1. a domain's $MSK$ cannot exist on a single machine.
2. only Alice should know her decryption key $SK_{Alice}$.

Much like Lightweight Signatures, our practical threat model does not defend against *active* mail server adversaries nor the total compromise of an end-user's personal computer. However, we do provide a reasonable privacy guarantee for users, using (primarily) the existing Internet infrastructure.

**Good Enough Security?** In certain limited cases, the unmodified deployment of Lightweight Signature keys may be good enough for encryption. It certainly achieves better privacy than the majority of users enjoy today. However, we can do better within the same deployment constraints.

## 1.4 Our (Two-Part) Solution

In this paper, we describe how to use the traditional approach of key splitting and distributed key generation [9] in our identity-based framework. We apply this technique in two ways, first to protect honest domains from single-attack compromise, then to protect honest users from overly curious incoming mail servers.

**Protecting Honest Domains.** A domain's $MSK$ can retroactively decrypt all messages encrypted against its public counterpart. Thus, we consider $MSK$ so valuable that it should never be stored on a single computer. However, it must remain functional enough to compute keys for new users on demand.

We propose that a domain maintain several servers that independently generate master key shares $(MPK_i, MSK_i)$. The master public key shares are combined into a single $MPK$ that is distributed via the DNS. Each server with $MSK_i$ individually sends Alice secret key share $SK_{Alice,i}$. Alice can then combine all shares into a single secret key $SK_{Alice}$. As expected, $SK_{Alice}$ correctly decrypts ciphertexts computed against $MPK$ and Alice's identity string $id\_string$.

**Protecting Honest Users.** Even with $MSK$ split amongst multiple servers, Alice's secret key shares can be intercepted by her *passive*, incoming mail server. Effectively, Lightweight PKI is insufficient for encryption because it provides a single, imperfect communication channel: email is decrypted using the single key $SK_{Alice}^{\texttt{wonderland.com}}$ issued by the master domain for `wonderland.com`, which is known to both Alice and her incoming mail server.

Our solution is thus to set up multiple channels, all of which are necessary to perform decryption. Email-based identification can provide lightweight certification as long as it defines one of these channels. To ensure Alice's privacy, only she should have access to all channels simultaneously.

Alice can, in fact, create one of these channels on her own. She generates $(MPK^{Alice}, MSK^{Alice})$ using parameters compatible with $MPK^{\texttt{wonderland.com}}$ and publishes $MPK^{Alice}$ via the mechanism of her choice, e.g. her web page. Then, her complete decryption key is a combination of $SK_{Alice}^{\texttt{wonderland.com}}$ and $SK_{Alice}^{Alice}$. Notice that, although Alice generates one key share on her own, she does not need to obtain certification for it, since an active adversary who spoofs it will not have $SK_{Alice}^{\texttt{wonderland.com}}$ to read her email. We return to these issues in Section 3.3.

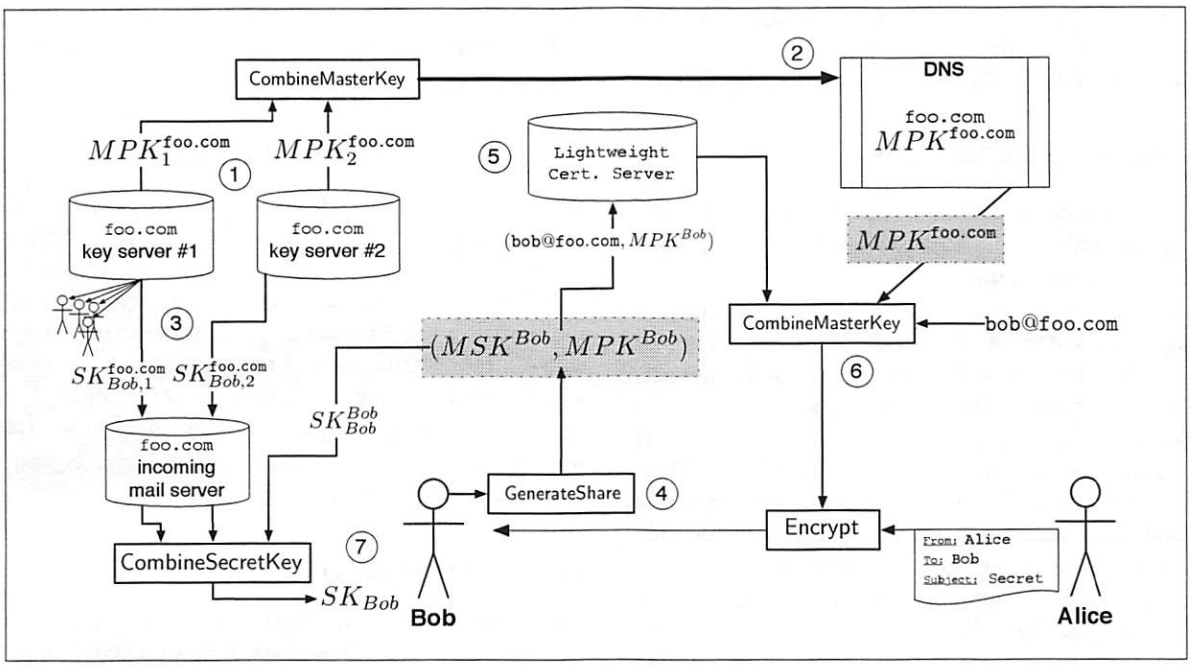



Figure 1: Lightweight Encryption: (1) The domain sets up two key servers, each in possession of a share of $MSK$, (2) The domain's shares of the master public key are combined into a single key $MPK^{\text{foo.com}}$ stored in the DNS, (3) each domain key server emails Bob a share of his secret key $SK_{Bob}^{\text{foo.com}}$ for that domain, (4) To achieve privacy from his mail server, Bob generates his own master keypair and stores his secret key $SK_{Bob}^{Bob}$, (5) Bob publishes his uncertified master public key $MPK^{Bob}$ via a keyserver or simple web page, (6) To encrypt to Bob, Alice retrieves his two master public keys and combines them, (7) To decrypt, Bob combines his three secret keys.

**Splitting and Combining Keys.** An honest domain is protected by splitting and recombining the $MSK$, while an honest user is protected by creating a new master key share and recombining it with the domain's $MPK$. In both cases, our technical contribution is the application of these key splitting-and-recombination techniques in the context of identity-based encryption. We review a known approach for the popular Boneh-Franklin IBE scheme [4] and describe, for the first time, such a technique for the new Waters IBE system [13] as well.

## 1.5 Related Work

Since the early days of encrypted email with S/MIME [10] and PGP [14], the public-key infrastructure requirement has been noted as hampering widespread deployment [3]. Numerous approaches have been proposed in the intervening years. Most recently, Garfinkel suggested viral public key distribution and key continuity [8] in order to leverage existing communication channels and reduce the need for a public-key infrastructure. Both of these techniques can be combined with our work to strengthen the overall system.

## 1.6 Roadmap

In Section 2, we review identity-based cryptography. In Section 3, we present Lightweight Encryption at a high level. The technical details of this solution are in Section 4. Finally, we briefly discuss some extensions and variations of our solution in Section 5, before concluding in Section 6.

# 2 Preliminaries

With identity-based cryptography, a user's public key is the combination of $MPK$ and $id\_string$, usually the user's email address. More specifically, an identity-based key management interface offers the following calls:

- Generate($key\_length$): outputs $(MPK, MSK)$, a master public key and corresponding master secret key of prescribed key length.
- ExtractSecretKey($MSK, id\_string$): outputs $SK$, the secret key that corresponds to the user $id\_string$ in domain $MPK$.
- VerifySecretKey($MPK, id\_string, SK$): outputs True only if $SK$ matches $id\_string$ in the master domain designated by $MPK$.

The identity-based encryption (IBE) interface is then:

- Encrypt($MPK, id\_string, m$): encrypts $m$ for user $id\_string$ in domain $MPK$.
- Decrypt($SK, c$): decrypts $c$ with secret key $SK$.

## 3 Lightweight Encryption

As previously discussed, there are two potential problems with applying the Lightweight PKI, in its basic form, to email encryption.

First, the compromise of an $MSK$ causes significant damage for all users of the domain. Thus, our first goal is to provide domains with more safeguards for protecting this key. Second, Alice's incoming mail server for domain `wonderland.com` knows all of Alice's secret cryptographic keys, specifically $SK_{Alice}^{\texttt{wonderland.com}}$. Thus, all of Alice's incoming encrypted mail can be decrypted and read by her mail server. Our second goal is to provide users with more privacy, such that no passive adversary, not even Alice's incoming mail server, can ever read encrypted email destined for Alice.

Both of these goals can be achieved by either splitting (first case) or combining (second case) the public or secret keys of IBE schemes. We begin by describing this functionality.

### 3.1 Key Splitting and Combining

We rework the identity-based function calls from Section 2 to enable: (1) combining stand-alone $MPK_i$'s into a single $MPK$, (2) combining stand-alone $SK_{id\_string,i}$'s into $SK_{id\_string}$, and (3) splitting the key generation of $MSK$ and the extraction of $SK_{id\_string}$ among many servers.

- GenerateShare($params$)
  outputs $(MPK_i, MSK_i)$, a share of a master public and corresponding master secret key.
- CombineMasterKey($MPK_1, \ldots, MPK_n$)
  outputs $MPK$, the master public key corresponding to the $n$ input public key shares.
- ExtractSecretShare($MSK_i, id\_string$)
  outputs $SK_i$, the user key share that corresponds to the user $id\_string$ and the master key share $MSK_i$.
- VerifySecretShare($MPK_i, id\_string, SK_i$)
  outputs True or False, depending on whether $SK_i$ corresponds to the secret key share for input $id\_string$ against master key share $MPK_i$.
- CombineSecretKey($SK_1, \ldots, SK_n$)
  outputs $SK$, the secret key corresponding to the $n$ input secret key shares.

We stress an important feature: a secret key share $SK_{id\_string,i}$ generated against a master share $MPK_i$ can function as a stand-alone key, with $MPK_i$ the stand-alone master. Each key share is a fully functional key, if need be. This feature enables the reverse operation of key recombination from existing, independent, fully-functional keys.

### 3.2 Protocol for Email Domains

Using the functionality from Section 3.1, a given email domain can use multiple servers to manage its master keypair. These servers can all be online to send Alice her secret key shares, yet any adversary wanting to hijack the domain must hack into *every* server to reconstruct $MSK$. Individuals wishing to send Alice an encrypted email can retrieve the pre-combined $MPK$ through the DNS, against which they can form an encryption. They need not know that the corresponding $MSK$ is shared among multiple servers.

### 3.3 Protocol for Users

Alice, with address `alice@wonderland.com`, can use functionality from Section 3.1 to prevent her mail server from reading her incoming emails as follows:

1. generate a fresh master keypair ($MPK^{Alice}$, $MSK^{Alice}$) using the same *params* as those of `wonderland.com`.
2. distribute (`alice@wonderland.com`, $MPK^{Alice}$) via a keyserver, web site, etc. The association between `alice@wonderland.com` and $MPK^{Alice}$ need not be certified.
3. generate a secret key complement $SK_{Alice}^{Alice}$ using the string `alice@wonderland.com` and $MSK^{Alice}$. Run CombineSecretKey on $SK_{Alice}^{\texttt{wonderland.com}}$, from her domain, and the newly created $SK_{Alice}^{Alice}$ to obtain a single secret key $SK_{Alice}$.

Then, when Bob wishes to send Alice an encrypted email at address `alice@wonderland.com`, he performs the following actions:

1. obtain $MPK^{\texttt{wonderland.com}}$ using the established DNS-based mechanism of the Lightweight PKI.
2. obtain $MPK^{Alice}$ from a keyserver or Alice's web site.
3. combine the two master public keys: $MPK$ = CombineMasterKey($MPK^{\texttt{alice@wonderland.com}}$, $MPK^{Alice}$).
4. encrypt a message $m$ for Alice: Encrypt($MPK$, `alice@wonderland.com`, $m$).

With this approach, Alice prevents her mail server from decrypting and reading her email with only $SK_{Alice}^{\texttt{wonderland.com}}$. Even an adversary (other than Alice's mail server) who spoofs Alice's uncertified key

$MPK^{Alice}$ *cannot* decrypt and read her email. Our threat model does not protect against a mail server that spoofs its own user's webpage or keyserver entry, as this type of attack is unlikely in practice.

However, an active attacker – other than Alice's mail server – could mount a denial-of-service attack against Alice by publishing spurious master public keys associated with Alice's email address. We suggest methods for preventing this attack in Section 5.

### 3.4 Adoption and Deployment

Lightweight encryption offers flexible deployment options. Privacy increases with each domain deployment of a Lightweight PKI, and even naive users get some privacy. We explore two usage cases:

**Scenario One: Naive Users.** Using the Lightweight PKI, non-savvy users can use an updated mail client which automatically encrypts their outgoing mail and decrypts their incoming mail. To encrypt to `bob@foo.com`, Alice's client simply needs to obtain $MPK^{\texttt{foo.com}}$ via DNS. Recall that even if this domain splits its master secret key among many servers to safeguard against hackers, it need only post this combined key in the DNS. Decryption is even easier. The mail server for `foo.com` emails Bob his secret key $SK_{Bob}^{\texttt{foo.com}}$. Bob's email client can transparently recognize and process such emails, then using the included key to decrypt the email from Alice.

**Scenario Two: Advanced Users.** Suppose Alice and Bob are more advanced users, able to follow the protocol in Section 3.3. Bob publishes an additional public key on his web site. Alice combines this public key with the one defined by $MPK^{\texttt{foo.com}}$. Her email to Bob is then encrypted such that even Bob's incoming mail server is unable to decrypt it: only Bob can. Note that Bob never needs to certify any of this additional key material.

## 4 How To Split IBE Master Keys

We present methods for splitting, among a group of trustees, the key generation and verification algorithms for two predominant IBE systems. Both key types [4, 13] are based on bilinear maps and support efficient identity-based encryption and signature schemes [1].

### 4.1 Bilinear Maps

Let BM_Setup be an algorithm that, on input the security parameter $1^k$, outputs $(q, g, h, G_1, G_2, e)$, where $e$ is a function mapping $G_1 \times G_1$ to $G_2$, where both $G_1$ and $G_2$ are groups of prime order $q = \Theta(2^k)$, and elements $g$ and $h$ both generate $G_1$. The function $e$ has the properties [4]: (*Bilinear*) for all $g, h \in G_1$, for all $a, b \in \mathbb{Z}_q$, $e(g^a, h^b) = e(g, h)^{ab}$; (*Non-degenerate*) if $g$ is a generator of $G_1$, then $e(g, g)$ generates $G_2$; and (*Efficient*) computing $e(g, h)$ is efficient for all $g, h \in G_1$.

### 4.2 Waters Key Pairs

We present these algorithms for (a reformulation of) the Waters key pairs [13, 1] when *all trustees are assumed to be honest*. This is the case, for example, when a company wants to split its $MSK$ among a number of servers that it owns and operates in order to make stealing the $MSK$ more difficult for hackers. In Section 4.4, we address the possibility of malicious servers.

**Single Server Key Algorithms**

- Generate$(1^k)$ outputs $MPK = (params, g^b)$ and $MSK = b$ for $params = (q, g, h, G_1, G_2, e, H)$ where $H$ is a function mapping strings in $\{0,1\}^k$ to elements in $G_1$ and the remaining parameters are generated by running BM_Setup$(1^k)$. We assume that the discrete logarithm of $h$ with respect to $g$ is unknown. (Here, $H$ is a particular implementation of a hash function, not a generic random oracle. We defer to Waters for the details [13].)

- ExtractSecretKey$(MSK, id_string)$ outputs the user secret key $SK = (h^b H(id\_string)^r, g^r)$, for a random $r \in \mathbb{Z}_q$.

- VerifySecretKey$(MPK, id_string, SK)$ parses $SK$ as $(A, B)$, outputs True if and only if $e(A, g)/e(B, H(id\_string)) = e(h, g^b)$.

**Multiple Server Key Algorithms**

- Setup$(1^k)$: $params = (q, g, h, G_1, G_2, e, H)$.

- GenerateShare$(params)$ outputs a master key share $MPK_i = g^{b_i}$ and the corresponding secret key share $MSK_i = b_i$.

- CombineMasterKey$(MPK_1, .., MPK_n)$ outputs

$$MPK = \prod_{i=1}^{n} g^{b_i} = g^b.$$

- ExtractSecretShare$(MSK_i, id_string)$ outputs

$$SK_i = (h^{b_i} H(id\_string)^{r_i}, g^{r_i})$$

for a random $r_i \in \mathbb{Z}_q$.

- VerifySecretShare($MPK_i, id\_string, SK_i$) parses $SK_i$ as $(A_i, B_i)$, outputs True only if

$$e(A_i, g)/e(B_i, H(id\_string)) = e(h, g^{b_i}).$$

- CombineSecretKey($SK_1, \ldots, SK_n$) outputs

$$\begin{aligned} SK &= \left(\prod_{i=1}^{n} h^{b_i} H(ID)^{r_i}, \prod_{i=1}^{n} g^{r_i}\right) \\ &= \left(h^b H(ID)^r, g^r\right). \end{aligned}$$

### 4.3 Boneh-Franklin Key Pairs

Boneh and Franklin [4] briefly discuss how the key pairs for their encryption scheme can be generated and verified in a distributed fashion. The key pairs are simpler than the above Waters scheme. The master key pairs are of the form $(MPK, MSK) = (g^s, s)$ and user key pairs of the form $(PK, SK) = (H(id\_string), H(id\_string)^s)$, where $H : \{0,1\}^* \rightarrow G_1$ is a hash function. For the remaining details, we defer to the Boneh and Franklin [4].

### 4.4 Dealing with Untrusted Servers

There are two ways that a malicious server (or servers) can undermine the security of the previous schemes.

**Key Generation Issues.** It is possible for a set of colluding servers to deviate from the basic GenerateShare protocols above and choose their shares $MPK_i$ in such a way as to bias the final master key $MPK$. An adversary might post a malicious second-channel key $MPK^{evil}$ which "cancels out" Alice's first-channel key from $MPK^{\texttt{wonderland.com}}$, making the combined $SK_{Alice}$ a known value.

We can prevent these attacks by requiring that each server posting an $MPK$ also posts a proof that they know the corresponding value $MSK$ *without revealing* $MSK$. This can be done by standard cryptographic techniques outside the scope of this paper.

**Threshold Issues.** After a set of $n$ servers have published a final master key $MPK$, one or more malicious servers may refuse to provide Alice with their shares of her user secret key $SK_{Alice}$. It would be better if some subset of the servers, say any group of $k+1$ out of $n$ for $n/2 \leq k < n$, can provide enough secret shares to reconstruct $SK_{Alice}$ for Alice. This safeguard can be achieved by a direct application of techniques due to Gennaro et al. [9] *when the majority of servers are honest*, as previously noted for Boneh-Franklin key pairs [4].

## 5 Extensions and Variations

**Stronger Lightweight Certificates.** In Section 3, Alice creates her second encryption channel by posting $MPK^{Alice}$. An active adversary might post spurious $MPK^{evil}$ in order to hijack this second channel. Since the adversary does not know the secret key $SK_{Alice}^{\texttt{wonderland.com}}$ for the first channel, the best he can achieve is a denial-of-service attack. In practice, we want to prevent such attacks too. Alice can do so by using her existing lightweight signing key $SK_{Alice}^{\texttt{wonderland.com}}$ to sign her second-channel $MPK^{Alice}$. If Alice's mail server were actively malicious, it could spoof this signed $MPK^{\texttt{wonderland.com}}$. However, as stated earlier, our threat model does not account for this (unlikely) attack.

**Double Email-Based Identification.** Lightweight encryption works because Alice has two encryption channels: one from her email domain, and one from herself. As we saw in Section 4, the two keys from these channels can be combined to allow for a single encryption by the sender and a single decryption by the recipient.

Another way to set up two encryption keys is to have Alice advertise two email addresses at different domains, `alice@wonderland.com` and `alice@school.edu`. As the two domains `wonderland.com` and `school.edu` are unlikely to be using the same key parameters (e.g. the same bilinear map in Boneh-Franklin or Waters), the two secret keys obtained from each domain cannot be recombined. However, the sender can simply use double-encryption: encrypt the message first against $MPK^{\texttt{wonderland.com}}$, then encrypt that ciphertext against $MPK^{\texttt{school.edu}}$. Neither incoming mail server can decrypt Alice's email single-handedly, yet Alice can perform two decryptions to recover the message. (Note that this only provides added security if Alice isn't simply forwarding her email from `alice@school.edu` to `alice@wonderland.com`. She must log in to each incoming mail server independently.)

## 6 Conclusion

We introduced Lightweight Encryption as a means of realizing email encryption with a realistic adoption and deployment process. Certainly, more work is necessary, including user interface considerations, the widespread distribution of secondary user private keys, and the real-world validation of our deployment ideas.

Authentic, private email can help improve the quality of all Internet-based communication. Lightweight encryption provides a solid building block towards making email encryption practical.

**Acknowledgments.** Susan Hohenberger's work was supported by an NDSEG Fellowship.

## References

[1] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Ad-Hoc Group Signatures (including ID-based ones) from Almost Any Collection of Key Pairs, 2005. Available at http://theory.lcs.mit.edu/~rivest/publications.html.

[2] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Lightweight Signatures for Email, 2005. Available at http://theory.lcs.mit.edu/~rivest/publications.html.

[3] Steven M. Bellovin. Cryptography and the internet. In *CRYPTO*, volume 1462 of LNCS, pages 46–55, 1998.

[4] Dan Boneh and Matt Franklin. Identity-based Encryption from the Weil Pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[5] Ronald Cramer and Victor Shoup. A pratical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, volume 1642 of LNCS, pages 13–25, 1998.

[6] D. Eastlake. RFC 2535: Domain Name System Security Extensions, March 1999.

[7] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO '84*, pages 10–18, 1984.

[8] Simson L. Garfinkel. *Design Principles and Patterns for Computer Systems that are Simultaneously Secure and Usable*. PhD thesis, MIT, April 2005.

[9] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT*, volume 1592 of LNCS, pages 295–310, 1999.

[10] IETF. S/MIME Working Group. http://www.imc.org/ietf-smime/index.html.

[11] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining dig. signatures and public-key cryptosystems. *Com. of the ACM*, 21,2:120–126, 1978.

[12] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, volume 196, pages 47–53, 1984.

[13] Brent Waters. Efficient Identity-Based Encryption Without Random Oracles. In *EUROCRYPT*, volume 3494 of LNCS, pages 114–127, 2005.

[14] Phil Zimmerman. Pretty Good Privacy. http://www.pgp.com.